

---

# **Lease User Manual**

***Release 1.99.14.20260521081756.d9d7a78607***

**Banu Systems Private Limited**

**May 21, 2026**



# CONTENTS

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Hardware requirements . . . . .	3
1.2	Supported platforms . . . . .	3
1.3	RPM platforms . . . . .	4
1.4	Debian and Ubuntu platforms . . . . .	5
1.5	Upgrading . . . . .	5
<b>2</b>	<b>Programs</b>	<b>7</b>
2.1	<b>dhcpcd</b> --- DHCP server . . . . .	7
2.1.1	Synopsis . . . . .	7
2.1.2	Description . . . . .	7
2.1.3	Operation . . . . .	7
2.1.4	Command line . . . . .	8
2.1.5	Options . . . . .	9
2.1.6	Loopback interfaces . . . . .	11
2.1.7	UDP and TCP ports . . . . .	11
2.1.8	Configuration . . . . .	12
2.1.8.1	Subnets . . . . .	12
2.1.8.2	Lease time . . . . .	13
2.1.8.3	BOOTP support . . . . .	13
2.1.8.4	Options . . . . .	14
2.1.9	OMAPI . . . . .	14
2.1.9.1	The lease object . . . . .	15
2.1.9.2	The host object . . . . .	17
2.1.9.3	The group object . . . . .	18
2.1.9.4	The control object . . . . .	18
2.1.9.5	The failover-state object . . . . .	18
2.1.10	Signals . . . . .	21
2.1.11	Files . . . . .	21
2.1.12	See also . . . . .	21
2.1.13	Copyright . . . . .	22
2.2	<b>dhclient</b> --- DHCP client . . . . .	22
2.2.1	Synopsis . . . . .	22
2.2.2	Description . . . . .	22
2.2.3	Operation . . . . .	22
2.2.4	Command line . . . . .	23

2.2.5	Options . . . . .	23
2.2.5.1	Options available for DHCPv6 only . . . . .	26
2.2.5.2	Modifying default file locations . . . . .	27
2.2.6	UDP and TCP ports . . . . .	28
2.2.7	Configuration . . . . .	28
2.2.8	OMAPI . . . . .	28
2.2.8.1	The control object . . . . .	29
2.2.9	Environment variables . . . . .	29
2.2.10	Files . . . . .	30
2.2.11	See also . . . . .	30
2.2.12	Copyright . . . . .	30
2.3	<b>dhclient-script</b> --- DHCP client network configuration script . . .	30
2.3.1	Description . . . . .	30
2.3.2	Hooks . . . . .	31
2.3.3	Operation . . . . .	31
2.3.3.1	MEDIUM . . . . .	31
2.3.3.2	PREINIT . . . . .	32
2.3.3.3	BOUND . . . . .	32
2.3.3.4	RENEW . . . . .	33
2.3.3.5	REBIND . . . . .	33
2.3.3.6	REBOOT . . . . .	33
2.3.3.7	EXPIRE . . . . .	33
2.3.3.8	FAIL . . . . .	33
2.3.3.9	STOP . . . . .	33
2.3.3.10	RELEASE . . . . .	34
2.3.3.11	NBI . . . . .	34
2.3.3.12	TIMEOUT . . . . .	34
2.3.4	Bugs . . . . .	34
2.3.5	See also . . . . .	34
2.3.6	Copyright . . . . .	35
2.4	<b>dhcrelay</b> - DHCP relay agent . . . . .	35
2.4.1	Synopsis . . . . .	35
2.4.2	Description . . . . .	35
2.4.3	Operation . . . . .	35
2.4.4	Options . . . . .	36
2.4.4.1	Protocol selection . . . . .	36
2.4.4.2	Specifying DHCPv4/BOOTP servers . . . . .	36
2.4.4.3	Options available for both DHCPv4 and DHCPv6 . . .	36
2.4.4.4	Options available for DHCPv4 only . . . . .	37
2.4.4.5	Options available for DHCPv6 only . . . . .	39
2.4.5	See also . . . . .	39
2.4.6	Bugs . . . . .	39
2.4.7	Copyright . . . . .	40
2.5	<b>omshell</b> --- OMAPI command shell . . . . .	40
2.5.1	Synopsis . . . . .	40
2.5.2	Description . . . . .	40
2.5.3	Options . . . . .	40
2.5.4	Local and remote objects . . . . .	40

2.5.5	Opening a connection . . . . .	41
2.5.6	Creating local objects . . . . .	41
2.5.7	Associating local and remote objects . . . . .	42
2.5.8	Viewing a remote object . . . . .	42
2.5.9	Modifying a remote object . . . . .	43
2.5.10	New remote objects . . . . .	44
2.5.11	Resetting attributes . . . . .	45
2.5.12	Refreshing objects . . . . .	45
2.5.13	Deleting objects . . . . .	45
2.5.14	Help . . . . .	46
2.5.15	See also . . . . .	46
2.5.16	Copyright . . . . .	46
<b>3</b>	<b>Configuration and data</b>	<b>47</b>
3.1	<code>dhcpcd.conf</code> --- DHCP server configuration . . . . .	47
3.1.1	Description . . . . .	47
3.1.2	Examples . . . . .	49
3.1.3	Address pools . . . . .	51
3.1.4	Dynamic address allocation . . . . .	52
3.1.5	IP address conflict prevention . . . . .	53
3.1.6	DHCP failover . . . . .	53
3.1.6.1	Failover startup . . . . .	54
3.1.6.2	Configuring failover . . . . .	55
3.1.7	Client classing . . . . .	61
3.1.7.1	Subclasses . . . . .	61
3.1.7.2	Per-class limits on dynamic address allocation . . . . .	63
3.1.7.3	Spawning classes . . . . .	63
3.1.7.4	Combining match, match if, and spawn with . . . . .	64
3.1.8	Dynamic DNS updates . . . . .	64
3.1.9	The DNS UPDATE scheme . . . . .	64
3.1.9.1	Dynamic DNS UPDATE security . . . . .	67
3.1.10	Events . . . . .	69
3.1.11	Declarations . . . . .	69
3.1.12	Allow and deny . . . . .	72
3.1.12.1	allow, deny, and ignore in scope . . . . .	73
3.1.12.2	allow and deny within pool declarations . . . . .	76
3.1.13	Parameters . . . . .	78
3.1.14	Setting parameter values using expressions . . . . .	97
3.1.15	Reserved leases . . . . .	98
3.1.16	References . . . . .	98
3.1.17	Files . . . . .	99
3.1.18	See also . . . . .	99
3.1.19	Copyright . . . . .	99
3.2	<code>dhcpcd.leases</code> --- DHCP lease database . . . . .	99
3.2.1	Description . . . . .	99
3.2.2	Format . . . . .	100
3.2.3	Common statements for lease declarations . . . . .	100
3.2.3.1	Dates . . . . .	100

	3.2.3.2	General variables	101
	3.2.3.3	DDNS Variables	101
	3.2.4	Executable statements	102
	3.2.5	The DHCPv4 lease declaration	102
	3.2.6	The DHCPv6 lease (IA) declaration	105
	3.2.7	The failover peer state declaration	106
	3.2.8	Files	106
	3.2.9	See also	106
	3.2.10	Copyright	106
3.3		<code>dhclient.conf</code> --- DHCP client configuration	107
	3.3.1	Description	107
	3.3.2	Protocol timing	107
	3.3.3	DHCPv6 lease selection	109
	3.3.4	Lease requirements and requests	109
	3.3.5	Dynamic DNS updates	110
	3.3.6	Option modifiers	111
	3.3.7	Lease declarations	112
	3.3.8	Alias declarations	115
	3.3.9	Other declarations	115
	3.3.10	Example	118
	3.3.11	Files	118
	3.3.12	See also	119
	3.3.13	Copyright	119
3.4		<code>dhclient.leases</code> --- DHCP client lease database	119
	3.4.1	Description	119
	3.4.2	Files	119
	3.4.3	See also	119
	3.4.4	Copyright	120
3.5		<code>dhcp-eval</code> --- DHCP conditional evaluation	120
	3.5.1	Description	120
	3.5.2	Conditional behavior	120
	3.5.2.1	The <code>if</code> statement	120
	3.5.2.2	The <code>switch</code> statement	121
	3.5.3	Boolean expressions	122
	3.5.4	Data expressions	123
	3.5.5	Numeric expressions	126
	3.5.6	Action expression	127
	3.5.7	Dynamic DNS updates	129
	3.5.8	See also	129
	3.5.9	Copyright	129
3.6		<code>dhcp-options</code> --- DHCP options	129
	3.6.1	Description	129
	3.6.2	Option statements	129
	3.6.3	Setting option values using expressions	130
	3.6.4	Standard DHCPv4 options	130
	3.6.5	Relay agent information option	145
	3.6.6	Client FQDN suboptions	146
	3.6.7	NetWare/IP suboptions	147

3.6.8	Standard DHCPv6 options . . . . .	148
3.6.9	Accessing DHCPv6 relay options . . . . .	155
3.6.10	Defining new options . . . . .	155
3.6.10.1	Boolean . . . . .	156
3.6.10.2	Integer . . . . .	156
3.6.10.3	IPv4 address . . . . .	156
3.6.10.4	IPv6 address . . . . .	157
3.6.10.5	Text . . . . .	157
3.6.10.6	Data string . . . . .	157
3.6.10.7	Domain list . . . . .	158
3.6.10.8	Encapsulation . . . . .	158
3.6.10.9	Arrays . . . . .	158
3.6.10.10	Records . . . . .	159
3.6.11	Vendor encapsulated options . . . . .	159
3.6.12	See also . . . . .	162
3.6.13	Copyright . . . . .	162
<b>4</b>	<b>Release notes</b>	<b>163</b>
4.1	Lease 1.99.14 . . . . .	163
4.2	Lease 1.99.13 . . . . .	165
4.3	Lease 1.99.12 . . . . .	166
4.4	Lease 1.99.11 . . . . .	166
4.5	Lease 1.99.10 . . . . .	166
4.6	Lease 1.99.9 . . . . .	167
4.7	Lease 1.99.8 . . . . .	168
4.8	Lease 1.99.7 . . . . .	169
4.9	Lease 1.99.6 . . . . .	170
4.10	Lease version numbering scheme . . . . .	170
4.10.1	Stable and development versions . . . . .	171
4.11	Lease branches . . . . .	171
4.12	History of Lease . . . . .	172
<b>5</b>	<b>License</b>	<b>173</b>
<b>6</b>	<b>Data and privacy</b>	<b>185</b>
	<b>Index</b>	<b>187</b>





**Warning:** This document is still a work-in-progress. Large parts of it may be inaccurate, and it may mention programs, features and configuration options that do not exist in Lease. This warning will be removed when the manual is known to be correct.



## INSTALLATION

### 1.1 Hardware requirements

Lease is packaged for a set of supported operating system platforms (see *Supported platforms*). Any machine (real or virtual) with one of these platforms can be used to run Lease.

DHCP hardware requirements have traditionally been quite modest. For many installations, servers that have been pensioned off from active duty can perform admirably as DHCP servers. For serving a small LAN, even low-performance machines may be sufficient. If the server's operational duties are larger, then a suitably performant machine can be selected.

We aren't able to recommend specifications in this document as it would be outdated quickly. It is best to profile the usage patterns and prepare a hardware configuration accordingly.

**Error:** TODO: Add a link to Lease support for help with hardware configuration.

**Error:** TODO: Add a link to a tuning section.

### 1.2 Supported platforms

Lease is written to run on POSIX operating systems. The following platforms are supported by this release of Lease:

- Red Hat Enterprise Linux 9 (x86\_64)
- Red Hat Enterprise Linux 9 (aarch64)
- Red Hat Enterprise Linux 10 (x86\_64)
- Red Hat Enterprise Linux 10 (aarch64)

- Fedora 43 (x86\_64)
- Fedora 43 (aarch64)
- Fedora 44 (x86\_64)
- Fedora 44 (aarch64)
- Debian 12 "bookworm" (amd64)
- Debian 12 "bookworm" (arm64)
- Debian 13 "trixie" (amd64)
- Debian 13 "trixie" (arm64)
- Debian 13 "trixie" (riscv64)
- Ubuntu 22.04 LTS "jammy" (amd64)
- Ubuntu 22.04 LTS "jammy" (arm64)
- Ubuntu 24.04 LTS "noble" (amd64)
- Ubuntu 24.04 LTS "noble" (arm64)
- Ubuntu 24.04 LTS "noble" (riscv64)

Packages for current versions of FreeBSD will be added in the future.

## 1.3 RPM platforms

---

**Note:** Users of AlmaLinux, Rocky Linux, and Oracle Linux distributions may use the packages for the corresponding Red Hat Enterprise Linux version.

---

Installation instructions for each supported platform will be available soon. For now, if you know how to install RPMs using **dnf**, please adapt the following instructions for your platform.

To install Lease on Red Hat Enterprise Linux 10 (x86\_64), you may run the following commands as the `root` user.

First, install the `akira-release` RPM package that will add the `akira-epel` and `akira-epel-testing` DNF repositories to your system, as well as associated GPG keys used to verify signed RPM packages from these repositories:

```
# dnf install https://download.banu.com/packages/akira/1.99/epel/  
→testing/10/x86_64/akira-release-1.99.14.20260521081756.d9d7a78607-  
→1.el10.noarch.rpm
```

Then, enable the `akira-epel-testing` DNF repository:

```
# dnf config-manager setopt akira-epel-testing.enabled=1
```

Then, install the `lease` RPM package that will install the Lease software and documentation:

```
# dnf install lease
```

Then, if you wish to run the DHCP servers, configure them suitably by editing `/etc/lease/dhcpd.conf` and `/etc/lease/dhcpd6.conf`, and then run them:

```
# systemctl enable --now dhcpd
# systemctl enable --now dhcp6
```

---

**Note:** For information about Lease's version numbering, see [Lease version numbering scheme](#). For information about Lease's branches and EOL dates, see [Lease branches](#).

---

## 1.4 Debian and Ubuntu platforms

Debian and Ubuntu packages are currently not published in `apt` repositories. If you would like to use them, please contact us.

## 1.5 Upgrading

Please read the release notes before upgrading to a newer version of Lease (see the chapter titled [Release notes](#)).



## PROGRAMS

The following sections document programs that are part of the Lease software distribution.

### 2.1 `dhcpcd` --- DHCP server

#### 2.1.1 Synopsis

```
dhcpcd [ -p <port> ] [ -nio <method> ] [ -f ] [ -d ] [ -q ] [ -t | -T ] [ -4 | -6 ] [ -4o6 <port> ] [ -s <server> ] [ -cf <config-file> ] [ -lf <lease-file> ] [ -pf <pid-file> ] [ --no-pid ] [ -user <user> ] [ -group <group> ] [ -chroot <dir> ] [ <if0> ] [ ... <ifN> ]  
dhcpcd [ -h | -V ]
```

#### 2.1.2 Description

**dhcpcd** implements the Dynamic Host Configuration Protocol (DHCP) and the Internet Bootstrap Protocol (BOOTP). DHCP allows hosts on a TCP/IP network to request and be assigned IP addresses, and also to discover information about the network to which they are attached. BOOTP provides similar functionality, with certain restrictions.

#### 2.1.3 Operation

The DHCP protocol allows a host which is unknown to the network administrator to be automatically assigned a new IP address out of a pool of IP addresses for its network. In order for this to work, the network administrator allocates address pools in each subnet and enters them into the `dhcpcd.conf(5)` file.

There are two versions of the DHCP protocol DHCPv4 and DHCPv6. At startup the server may be started for one or the other via the `-4` or `-6` arguments.

On startup, **dhcpcd** reads the `dhcpcd.conf(5)` file and stores a list of available addresses on each subnet in memory. When a client requests an address using the DHCP

protocol, **dhcpcd** allocates an address for it. Each client is assigned a lease, which expires after an amount of time chosen by the administrator (by default, one day). Before leases expire, the clients to which leases are assigned are expected to renew them in order to continue to use the addresses. Once a lease has expired, the client to which that lease was assigned is no longer permitted to use the leased IP address.

In order to keep track of leases across system reboots and server restarts, **dhcpcd** keeps a list of leases it has assigned in the `dhcpcd.leases(5)` file. Before **dhcpcd** grants a lease to a host, it records the lease in this file and makes sure that the contents of the file are flushed to disk. This ensures that even in the event of a system crash, **dhcpcd** will not forget about a lease that it has assigned. On startup, after reading the `dhcpcd.conf(5)` file, **dhcpcd** reads the `dhcpcd.leases(5)` file to refresh its memory about what leases have been assigned.

New leases are appended to the end of the `dhcpcd.leases(5)` file. In order to prevent the file from becoming arbitrarily large, from time to time **dhcpcd** creates a new `dhcpcd.leases(5)` file from its in-core lease database. Once this file has been written to disk, the old file is renamed `dhcpcd.leases~`, and the new file is renamed `dhcpcd.leases`. If the system crashes in the middle of this process, whichever `dhcpcd.leases(5)` file remains will contain all the lease information, so there is no need for a special crash recovery process.

BOOTP support is also provided by this server. Unlike DHCP, the BOOTP protocol does not provide a protocol for recovering dynamically-assigned addresses once they are no longer needed. It is still possible to dynamically assign addresses to BOOTP clients, but some administrative process for reclaiming addresses is required. By default, leases are granted to BOOTP clients in perpetuity, although the network administrator may set an earlier cutoff date or a shorter lease length for BOOTP leases if that makes sense.

BOOTP clients may also be served in the old standard way, which is to simply provide a declaration in the `dhcpcd.conf(5)` file for each BOOTP client, permanently assigning an address to each client.

Whenever changes are made to the `dhcpcd.conf(5)` file, **dhcpcd** must be restarted. To restart **dhcpcd**, send a SIGTERM signal to the process ID contained in `/var/run/lease/dhcpcd.pid`, and then re-invoke **dhcpcd**. Because the DHCP server database is not as lightweight as a BOOTP database, **dhcpcd** does not automatically restart itself when it sees a change to the `dhcpcd.conf(5)` file.

## 2.1.4 Command line

The names of the network interfaces on which **dhcpcd** should listen for broadcasts may be specified on the command line. This should be done on systems where **dhcpcd** is unable to identify non-broadcast interfaces, but should not be required on other systems. If no interface names are specified on the command line **dhcpcd** will identify all network interfaces which are up, eliminating non-broadcast interfaces if possible, and listen for DHCP broadcasts on each interface.



## 2.1.5 Options

**-4**

Run as a DHCP (IPv4) server. This is the default and cannot be combined with **-6**.

**-4**

Run as a DHCPv6 (IPv6) server. This cannot be combined with **-4**.

**-4o6** <port>

Participate in the DHCPv4 over DHCPv6 protocol specified by [RFC 7341](#). This associates a DHCPv4 and a DHCPv6 server to allow the v4 server to receive v4 requests that were encapsulated in a v6 packet. Communication between the two servers is done on a pair of UDP sockets bound to `::1 <port>` and `<port> + 1`. Both servers must be launched using the same `<port>` argument.

**-p** <port>

The UDP port number on which **dhcpcd** should listen. If unspecified **dhcpcd** uses the default port of 67. This is mostly useful for debugging purposes.

**-nio** <method>

Specify the network I/O method to use for DHCP.

By default, **dhcpcd** uses an appropriate default network I/O method suitable for the platform it is built for. This is currently a "raw" method such as LPF on Linux and BPF on BSDs respectively. In this case, it also uses a fallback datagram socket to perform unicast communications when it may need address resolution.

The raw network I/O methods were implemented as it was not previously possible to receive and specify interface information for broadcast datagrams when reading and writing from regular UDP v4 sockets, or even to send datagrams addressed to `INADDR_BROADCAST` correctly at all. However, **dhcpcd** is now able to do so on all its supported platforms using regular UDP v4 sockets. It can also be run entirely as a non-root user if a privileged port is not used.

The following methods are available:

- **lpf** (Linux packet filter): This method is available only on Linux and is the default on Linux platforms.
- **bpf** (Berkeley packet filter): This method is available only on FreeBSD and is the default on FreeBSD.
- **socket** (datagram socket API): This method is available on Linux and FreeBSD platforms.

**-s** <address>

Specify an address or host name to which **dhcpcd** should send replies rather than the broadcast address (255.255.255.255). This option is only supported in IPv4.

**-f**

Force **dhcpcd** to run as a foreground process instead of as a daemon in the background.

- d**  
Send log messages to the standard error descriptor. This can be useful for debugging, and also where a complete log of all DHCP activity must be kept but **syslogd** is not reliable or otherwise cannot be used. Normally, **dhcpcd** will log all output using the *syslog(3)* function with the log facility set to LOG\_DAEMON. Note that **-d** implies **-f** (i.e., the daemon will not fork itself into the background).
- h**  
Print program usage information and exit.
- q**  
Be quiet at startup. This suppresses the printing of the entire copyright message during startup.
- t**  
Test the configuration file. The server tests the *dhcpcd.conf(5)* file for correct syntax, but will not attempt to perform any network operations. This can be used to test a new configuration file automatically before installing it.
- T**  
Test the lease file. The server tests the *dhcpcd.leases(5)* file for correct syntax, but will not attempt to perform any network operations. This can be used to test a new lease file automatically before installing it.
- user** <user>  
*setuid(2)* to <user> after completing privileged operations, such as creating sockets that listen on privileged ports. This also causes the lease file to be owned by <user>.
- group** <group>  
*setgid(2)* to <group> after completing privileged operations such as creating sockets that listen on privileged ports. This also causes the lease file to be owned by <group>.
- chroot** <directory>  
*chroot(2)* to <directory>.
- Warning:** This may occur before or after reading the configuration files depending on whether *--early-chroot* is used.
- early-chroot**  
If *-chroot* is used, then perform the *chroot(2)* before reading the configuration files.
- v**  
Print program version and exit.
- cf** <config-file>  
Specify path to the *dhcpcd.conf(5)* configuration file.

---

**Note:** Because of the importance of using the same lease database at all times when running **dhcpcd** in production, it is suggested that this option be used for testing lease files or database files in a non-production environment.

---

**-lf** <lease-file>

Specify path to the *dhcpcd.leases(5)* leases file.

---

**Note:** Because of the importance of using the same lease database at all times when running **dhcpcd** in production, it is suggested that this option be used for testing lease files or database files in a non-production environment.

---

**-pf** <pid-file>

Specify path to a file where the **dhcpcd** process ID is written.

**--no-pid**

Disable writing process ID files. By default, **dhcpcd** will write a process ID file. If it is invoked with this option it will not check for an existing **dhcpcd** process.

## 2.1.6 Loopback interfaces

Serving **dhcpcd** on loopback interfaces such as `lo` and `lo0` is not recommended. While there is some support for it in **dhcpcd**, note that loopback interfaces are not considered broadcast-capable on some underlying operating systems. Implementation of broadcast on loopback interfaces varies by operating system.

There are alternate virtual Ethernet-like interfaces such as `veth` on Linux and `epair` on FreeBSD which may serve your purpose for local use.

## 2.1.7 UDP and TCP ports

During operations the server may use multiple UDP and TCP ports to provide different functions. Which ports are used depends on the configuration in use. The following should provide an idea of what ports may be used.

When using LPF or BPF network I/O, a DHCPv4 server will open a raw UDP socket to receive and send most DHCPv4 packets. It also opens a fallback UDP socket for use in sending unicast packets. Normally these will both use the well known port number for BOOTPS.

For each DHCPv4 failover peer listed in the configuration file there will be a TCP socket listening for connections on the ports specified in the configuration file. When the peer connects, there will be another socket for the established connection. For the established connection the side (primary or secondary) opening the connection will use a random port.

For DHCPv6 the server opens a UDP socket on the well known `dhcpv6-server` port.

The server opens an ICMP socket for doing ICMP Echo (*ping(1)*) requests to check if addresses are in use.

If there is an `omapi-port` statement in the configuration file, then the server will open a TCP socket on that port to listen for OMAPI connections. When something connects another port will be used for the established connection.

When DNS UPDATE is used, the server will open a v4 and a v6 UDP socket on random ports. If the server is configured not to do DNS UPDATES (`ddns-update-style` set to `none` in the configuration file), the ports will never be opened.

## 2.1.8 Configuration

The syntax of the *dhcpcd.conf(5)* file is discussed separately. This section should be used as an overview of the configuration process, and the *dhcpcd.conf(5)* documentation should be consulted for detailed reference information.

### 2.1.8.1 Subnets

**dhcpcd** needs to know the subnet numbers and netmasks of all subnets for which it will be providing service. In addition, in order to dynamically allocate addresses, it must be assigned one or more ranges of addresses on each subnet which it can in turn assign to client hosts as they boot. Thus, a very simple configuration providing DHCP support might look like this:

```
subnet 239.252.197.0 netmask 255.255.255.0 {  
    range 239.252.197.10 239.252.197.250;  
}
```

Multiple address ranges may be specified like this:

```
subnet 239.252.197.0 netmask 255.255.255.0 {  
    range 239.252.197.10 239.252.197.107;  
    range 239.252.197.113 239.252.197.250;  
}
```

If a subnet will only be provided with BOOTP service and no dynamic address assignment, the `range` clause can be left out entirely, but the `subnet` statement must appear.

### 2.1.8.2 Lease time

DHCP leases can be assigned almost any duration from 0 seconds to infinity. What lease duration makes sense for any given subnet, or for any given installation, will vary depending on the kinds of hosts being served.

For example, in an office environment where systems are added from time to time and removed from time to time, but move relatively infrequently, it might make sense to allow lease durations of a month or more. In a final test environment on a manufacturing floor, it may make more sense to assign a maximum lease duration of 30 minutes --- enough time to go through a simple test procedure on a network appliance before packaging it up for delivery.

It is possible to specify two lease durations: the default duration that will be assigned if a client doesn't ask for any particular lease duration, and a maximum lease duration. These are specified as clauses to the `subnet` command:

```
subnet 239.252.197.0 netmask 255.255.255.0 {  
    range 239.252.197.10 239.252.197.107;  
    default-lease-time 600;  
    max-lease-time 7200;  
}
```

This particular `subnet` declaration specifies a default lease time of 600 seconds (ten minutes), and a maximum lease time of 7200 seconds (two hours). Other common values are 86400 (one day), 604800 (one week) and 2592000 (30 days).

Each subnet need not have the same lease --- in the case of an office environment and a manufacturing environment served by the same DHCP server, it might make sense to have widely disparate values for default and maximum lease times on each subnet.

### 2.1.8.3 BOOTP support

Each BOOTP client must be explicitly declared in the `dhcpd.conf(5)` file. A very basic client declaration will specify the client network interface's hardware address and the IP address to assign to that client. If the client needs to be able to load a boot file from the server, that file's name must be specified. A simple BOOTP client declaration might look like this:

```
host haagen {  
    hardware ethernet 08:00:2b:4c:59:23;  
    fixed-address 239.252.197.9;  
    filename "/tftpboot/haagen.boot";  
}
```

### 2.1.8.4 Options

DHCP (and also BOOTP with Vendor Extensions) provide a mechanism whereby the server can provide the client with information about how to configure its network interface (e.g., subnet mask), and also how the client can access various network services (e.g., DNS, IP routers, and so on).

These options can be specified on a per-subnet basis, and for BOOTP clients, also on a per-client basis. In the event that a BOOTP client declaration specifies options that are also specified in its subnet declaration, the options specified in the client declaration take precedence. A reasonably complete DHCP configuration might look something like this:

```
subnet 239.252.197.0 netmask 255.255.255.0 {  
    range 239.252.197.10 239.252.197.250;  
    default-lease-time 600 max-lease-time 7200;  
    option subnet-mask 255.255.255.0;  
    option broadcast-address 239.252.197.255;  
    option routers 239.252.197.1;  
    option domain-name-servers 239.252.197.2, 239.252.197.3;  
    option domain-name "example.org";  
}
```

A BOOTP host on that subnet that needs to be in a different domain and use a different DNS nameserver might be declared as follows:

```
host haagen {  
    hardware ethernet 08:00:2b:4c:59:23;  
    fixed-address 239.252.197.9;  
    filename "/tftpboot/haagen.boot";  
    option domain-name-servers 192.5.5.1;  
    option domain-name "example.com";  
}
```

A more complete description of the configuration file syntax is provided in *dhcpcd.conf*(5). A list of DHCP options are provided in *dhcp-options*(5).

### 2.1.9 OMAPI

**dhcpcd** provides the capability to modify some of its configuration while it is running, without stopping it, modifying its database files, and restarting it. This capability is currently provided using OMAPI (Object Management Application Programming Interface) --- an API for manipulating remote objects. OMAPI clients connect to the **dhcpcd** process using TCP/IP, authenticate, and can then examine the server's current status and make changes to it.

Rather than implementing the underlying OMAPI protocol directly, user programs should use the *dhcpcctl*(5) API or *omapi*(5) itself. *dhcpcctl*(5) is a wrapper that handles some of the housekeeping chores that OMAPI does not do automatically.

**Warning:** The above paragraph about *dhcpcctl(5)* and *omapi(5)* should be replaced with a discussion of using *omshell(1)*.

OMAPI exports objects, which can then be examined and modified. The DHCP server exports the following objects: lease, host, failover-state and group. Each object has a number of methods that are provided: lookup, create, and destroy. In addition, it is possible to look at attributes that are stored on objects, and in some cases to modify those attributes.

*omshell(1)* is a program that provides an interactive way to connect to, query, and possibly change **dhcpcd**'s state via OMAPI.

**Warning:** This section has to be rewritten, or moved to the developer documentation.

### 2.1.9.1 The lease object

Leases can't currently be created or destroyed, but they can be looked up to examine and modify their state.

Leases have the following attributes:

**state** <integer> [lookup, examine]

- 1 = free
- 2 = active
- 3 = expired
- 4 = released
- 5 = abandoned
- 6 = reset
- 7 = backup
- 8 = reserved
- 9 = bootp

**ip-address** <data> [lookup, examine]

The IP address of the lease.

**dhcp-client-identifier** <data> [lookup, examine, update]

The client identifier that the client used when it acquired the lease. Not all clients send client identifiers, so this may be empty.

**client-hostname** <data> [examine, update]

The value the client sent in the host-name option.

**host** <*handle*> [examine]

The host declaration associated with this lease, if any.

**subnet** <*handle*> [examine]

The subnet object associated with this lease (the subnet object is not currently supported).

**pool** <*handle*> [examine]

The pool object associated with this lease (the pool object is not currently supported).

**billing-class** <*handle*> [examine]

The handle to the class to which this lease is currently billed, if any (the class object is not currently supported).

**hardware-address** <*data*> [examine, update]

The hardware address (chaddr) field sent by the client when it acquired its lease.

**hardware-type** <*integer*> [examine, update]

The type of the network interface that the client reported when it acquired its lease.

**ends** <*time*> [examine]

The time when the lease's current state ends, as understood by the client.

**tsfp** <*time*> [examine]

The time when the lease's current state ends, as understood by the server.

**tsfp** <*time*> [examine]

The adjusted time when the lease's current state ends, as understood by the failover peer (if there is no failover peer, this value is undefined). Generally this value is only adjusted for expired, released, or reset leases while the server is operating in partner-down state, and otherwise is simply the value supplied by the peer.

**atsfp** <*time*> [examine]

The actual tsfp value sent from the peer. This value is forgotten when a lease binding state change is made, to facilitate retransmission logic.

**cltt** <*time*> [examine]

The time of the last transaction with the client on this lease.



### 2.1.9.2 The host object

Hosts can be created, destroyed, looked up, examined and modified. If a host declaration is created or deleted using OMAPI, that information will be recorded in the *dhcpcd.leases(5)* file. It is permissible to delete host declarations that are declared in the *dhcpcd.conf(5)* file.

Hosts have the following attributes:

**name** <data> [lookup, examine, modify]

The name of the host declaration. This name must be unique among all host declarations.

**group** <handle> [examine, modify]

The named group associated with the host declaration, if there is one.

**hardware-address** <data> [lookup, examine, modify]

The link-layer address that will be used to match the client, if any. Only valid if *hardware-type* is also present.

**hardware-type** <integer> [lookup, examine, modify]

The type of the network interface that will be used to match the client, if any. Only valid if *hardware-address* is also present.

**dhcp-client-identifier** <data> [lookup, examine, modify]

The dhcp-client-identifier option that will be used to match the client, if any.

**ip-address** <data> [examine, modify]

A fixed IP address which is reserved for a DHCP client that matches this host declaration. The IP address will only be assigned to the client if it is valid for the network segment to which the client is connected.

**statements** <data> [modify]

A list of statements in the format of the *dhcpcd.conf(5)* file that will be executed whenever a message from the client is being processed.

**known** <integer> [examine, modify]

If non-zero, indicates that a client matching this host declaration will be treated as known in pool permit lists. If zero, the client will not be treated as known.

### 2.1.9.3 The group object

Named groups can be created, destroyed, looked up, examined and modified. If a group declaration is created or deleted using OMAPI, that information will be recorded in the *dhcpcd.leases(5)* file. It is permissible to delete group declarations that are declared in the *dhcpcd.conf(5)* file.

Named groups currently can only be associated with hosts --- this allows one set of statements to be efficiently attached to more than one host declaration.

Groups have the following attributes:

**name** <data>

The name of the group. All groups that are created using OMAPI must have names, and the names must be unique among all groups.

**statements** <data>

A list of statements in the format of the *dhcpcd.conf(5)* file that will be executed whenever a message from a client whose host declaration references this group is processed.

### 2.1.9.4 The control object

The control object allows **dhcpcd** to be shutdown gracefully. If the server is doing failover with another peer, it will make a clean transition into the shutdown state and notify its peer, so that the peer can go into partner down, and then record the "recover" state in the lease file so that when the server is restarted, it will automatically resynchronize with its peer.

On shutdown the server will also attempt to cleanly shut down all OMAPI connections. If these connections do not go down cleanly after five seconds, they are shut down preemptively. It can take as much as 25 seconds from the beginning of the shutdown process to the time that the server actually exits.

To shut the server down, open its control object and set the state attribute to 2.

### 2.1.9.5 The failover-state object

The failover-state object is the object that tracks the state of the failover protocol as it is being managed for a given failover peer. The failover object has the following attributes (please see *dhcpcd.conf(5)* for descriptions of these attributes):

**name** <data> [examine]

Indicates the name of the failover peer relationship, as described in the server's configuration file.

**partner-address** <data> [examine]

Indicates the failover partner's IP address.

**local-address** <*data*> [examine]

Indicates the IP address that is being used by the DHCP server for this failover pair.

**partner-port** <*data*> [examine]

Indicates the TCP port on which the failover partner is listening for failover protocol connections.

**local-port** <*data*> [examine]

Indicates the TCP port on which the DHCP server is listening for failover protocol connections for this failover pair.

**max-outstanding-updates** <*integer*> [examine]

Indicates the number of updates that can be outstanding and unacknowledged at any given time, in this failover relationship.

**mclt** <*integer*> [examine]

Indicates the maximum client lead time in this failover relationship.

**load-balance-max-secs** <*integer*> [examine]

Indicates the maximum value for the secs field in a client request before load balancing is bypassed.

**load-balance-hba** <*data*> [examine]

Indicates the load balancing hash bucket array for this failover relationship.

**local-state** <*integer*> examine, modify

Indicates the present state of the DHCP server in this failover relationship. Possible values for state are:

- 1 - startup
- 2 - normal
- 3 - communications interrupted
- 4 - partner down
- 5 - potential conflict
- 6 - recover
- 7 - paused
- 8 - shutdown
- 9 - recover done
- 10 - resolution interrupted
- 11 - conflict done
- 254 - recover wait

In general it is not a good idea to make changes to this state. However, in the case that the failover partner is known to be down, it can be useful to set the DHCP server's failover state to partner down. At this point the DHCP server will take over service of the failover partner's leases as soon as possible, and will give out normal leases, not leases that are restricted by MCLT. If the DHCP server is put into the partner-down state when the other DHCP server is not in the partner-down state, but is not reachable, IP address assignment conflicts are possible, even likely. Once a server has been put into partner-down mode, its failover partner must not be brought back online until communication is possible between the two servers.

**partner-state** *<integer>* [examine]

Indicates the present state of the failover partner.

**local-stos** *<integer>* [examine]

Indicates the time at which the DHCP server entered its present state in this failover relationship.

**partner-stos** *<integer>* [examine]

Indicates the time at which the failover partner entered its present state.

**hierarchy** *<integer>* [examine]

Indicates whether the DHCP server is primary or secondary in this failover relationship.

Possible values are:

- 0 - primary
- 1 - secondary

**last-packet-sent** *<integer>* [examine]

Indicates the time at which the most recent failover packet was sent by this DHCP server to its failover partner.

**last-timestamp-received** *<integer>* [examine]

Indicates the timestamp that was on the failover message most recently received from the failover partner.

**skew** *<integer>* [examine]

Indicates the skew between the failover partner's clock and this DHCP server's clock.

**max-response-delay** *<integer>* [examine]

Indicates the time in seconds after which, if no message is received from the failover partner, the partner is assumed to be out of communication.

**cur-unacked-updates** *<integer>* [examine]

Indicates the number of update messages that have been received from the failover partner but not yet processed.

## 2.1.10 Signals

Certain UNIX signals cause **dhcpcd** to take specific actions:

**SIGTERM** Shuts down the DHCP server.

Signals can be sent using the *kill(1)* program. The result of sending any other signals to the server is undefined.

## 2.1.11 Files

*/etc/lease/dhcpd.conf*

The configuration file for the **dhcpcd** program. See *dhcpd.conf(5)* for more details.

*/var/lib/lease/dhcpd.leases*

The DHCP leases file. See *dhcpd.leases(5)* for more details.

*/var/lib/lease/dhcpd.leases~*

Old DHCP leases file.

*/var/lib/lease/dhcpd6.leases*

The DHCPv6 leases file. See *dhcpd.leases(5)* for more details.

*/var/lib/lease/dhcpd6.leases~*

Old DHCPv6 leases file.

*/var/run/lease/dhcpd.pid*

The default process ID file.

## 2.1.12 See also

*dhclient(8)*, *dhcrelay(8)*, *dhcpd.conf(5)*, *dhcp-options(5)*, *dhcpd.leases(5)*, *omshell(1)*

## 2.1.13 Copyright

Copyright (C) 2025-2026 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2004-2017 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1996-2003 by Internet Software Consortium.

## 2.2 `dhclient` --- DHCP client

### 2.2.1 Synopsis

```
dhclient [ -4 | -6 ] [ -S ] [ -N [ -N ... ] ] [ -T [ -T ... ] ] [ -P [ -P ... ] ] -R [ -i ] [ -I ] [ -4o6 <port> ] [ -D <LL|LLT> ] [ -p <port> ] [ -nio <method> ] [ -d ] [ -df <duid-lease-file> ] [ -e <VAR>*=*<value> ] [ -q ] [ -l ] [ -r | -x ] [ -lf <lease-file> ] [ -pf <pid-file> ] [ --no-pid ] [ -cf <config-file> ] [ -sf <script-file> ] [ -s <server-addr> ] [ -g <relay> ] [ -n ] [ -nw ] [ -w ] [ --dad-wait-time <seconds> ] [ -v ] [ <if0> [ ... <ifN> ] ]
```

```
dhclient [ -h | -V ]
```

### 2.2.2 Description

**dhclient** configures one or more network interfaces using the DHCP Protocol, BOOTP protocol, or if these protocols fail, by statically assigning an address.

### 2.2.3 Operation

The DHCP protocol allows a host to contact a central server which maintains a list of IP addresses which may be assigned on one or more subnets. A DHCP client may request an address from this pool, and then use it on a temporary basis for communication on network. The DHCP protocol also provides a mechanism whereby a client can learn important details about the network to which it is attached, such as the location of a default router, the location of a name server, and so on.

There are two versions of the DHCP protocol --- DHCPv4 and DHCPv6. At startup the client may be started for one or the other via the `-4` or `-6` options.

On startup, **dhclient** reads the `dhclient.conf(5)` for configuration instructions. It then gets a list of all the network interfaces that are configured in the current system. For each interface, it attempts to configure the interface using the DHCP protocol.

In order to keep track of leases across system reboots and server restarts, **dhclient** keeps a list of leases it has been assigned in the `dhclient.leases(5)` file. On startup, after reading the `dhclient.conf(5)` file, **dhclient** reads the `dhclient.leases(5)` file to refresh its memory about what leases it has been assigned.

When a new lease is acquired, it is appended to the end of the `dhclient.leases(5)` file. In order to prevent the file from becoming arbitrarily large, from time to

time **dhclient** creates a new *dhclient.leases(5)* file from its in-core lease database. The old version of the *dhclient.leases(5)* file is retained under the name *dhclient.leases~* until the next time **dhclient** rewrites the database.

Old leases are kept around in case the DHCP server is unavailable when **dhclient** is first invoked (generally during the initial system boot process). In that event, old leases from the *dhclient.leases(5)* file which have not yet expired are tested, and if they are determined to be valid, they are used until either they expire or until the DHCP server becomes available.

A mobile host which may sometimes need to access a network on which no DHCP server exists may be preloaded with a lease for a fixed address on that network. When all attempts to contact a DHCP server have failed, **dhclient** will try to validate the static lease, and if it succeeds, will use that lease until it is restarted.

A mobile host may also travel to some networks on which DHCP is not available but BOOTP is. In that case, it may be advantageous to arrange with the network administrator for an entry on the BOOTP database, so that the host can boot quickly on that network rather than cycling through the list of old leases.

## 2.2.4 Command line

The names of the network interfaces that **dhclient** should attempt to configure may be specified on the command line. If no interface names are specified on the command line **dhclient** will normally identify all network interfaces, eliminating non-broadcast interfaces if possible, and attempt to configure each interface.

It is also possible to specify interfaces by name in the *dhclient.conf(5)* file. If interfaces are specified in this way, then the client will only configure interfaces that are either specified in the configuration file or on the command line, and will ignore all other interfaces.

The client normally prints no output during its startup sequence. It can be made to emit verbose messages displaying the startup sequence events until it has acquired an address by supplying the *-v* command line argument. In either case, the client logs messages using the *syslog(3)* facility.

## 2.2.5 Options

**-4**

Use the DHCPv4 protocol to obtain an IPv4 address and configuration parameters. This is the default and cannot be combined with *-6*.

**-6**

Use the DHCPv6 protocol to obtain whatever IPv6 addresses are available along with configuration parameters. It cannot be combined with *-4*. The *-S*, *-T*, *-P*, *-N*, and *-D* arguments provide more control over aspects of the DHCPv6 processing.

---

**Note:** It is not recommended to mix queries of different types together or even to share the lease file between them.

---

**-4o6** <port>

Participate in the DHCPv4 over DHCPv6 protocol specified by [RFC 7341](#). This associates a DHCPv4 and a DHCPv6 client to allow the v4 client to send v4 requests encapsulated in a v6 packet. Communication between the two clients is done on a pair of UDP sockets bound to `::1 <port>` and `<port> + 1`. Both clients must be launched using the same `<port>` argument.

**-1**

Try to get a lease once. On failure exit with status 2. In DHCPv6 this sets the maximum duration of the initial exchange to timeout from `dhclient.conf(5)` with a default of 60 seconds.

**-d**

Force **dhclient** to run as a foreground process. Normally the DHCP client will run in the foreground until it has configured an interface at which time it will revert to running in the background.

**-h**

Print program usage information and exit.

**-nw**

Become a daemon immediately (nowait) rather than waiting until an IP address has been acquired.

**-q**

Be quiet at startup. This is the default.

**-v**

Enable verbose log messages.

**-w**

Continue running even if no broadcast interfaces were found. Normally **dhclient** will exit if it isn't able to identify any network interfaces to configure. On laptop computers and other computers with hot-swappable I/O buses, it is possible that a broadcast interface may be added after system startup. This argument can be used to cause the client not to exit when it doesn't find any such interfaces. The `omshell(1)` program can then be used to notify the client when a network interface has been added or removed, so that the client can attempt to configure an IP address on that interface.

**-n**

Do not configure any interfaces. This is most likely to be useful in combination with the `-w` argument.

**-e** <VAR>=<value>

Define additional environment variables for the environment in which `dhclient-script(8)` executes. Multiple `-e` arguments may be specified on the command line.



**-r**

Release the current lease and stop the running **dhclient** process as previously recorded in the PID file. When shutdown via this method, *dhclient-script*(8) will be executed with the specific reason for calling the script set.

---

**Note:** The client normally doesn't release the current lease as this is not required by the DHCP protocol but some cable ISPs require their clients to notify the server if they wish to release an assigned IP address.

---

**-x**

Without releasing the current lease, stop the running **dhclient** process as previously recorded in the PID file. When shutdown via this method *dhclient-script*(8) will be executed with the specific reason for calling the script set.

**-p** <port>

The UDP port number on which **dhclient** should listen and transmit. If unspecified, **dhclient** uses the default port of 68. This is mostly useful for debugging purposes. If a different port is specified on which the client should listen and transmit, the client will also use a different destination port --- one less than the specified <port>.

**-nio** <method>

Specify the network I/O method to use for DHCP.

By default, **dhclient** uses an appropriate default network I/O method suitable for the platform it is built for. This is currently a "raw" method such as LPF on Linux and BPF on BSDs respectively. In this case, it also uses a fallback datagram socket to perform unicast communications when it may need address resolution.

The raw network I/O methods were implemented as it was not previously possible to receive and specify interface information for broadcast datagrams when reading and writing from regular UDP v4 sockets, or even to send datagrams addressed to INADDR\_BROADCAST correctly at all. However, **dhclient** is now able to do so on all its supported platforms using regular UDP v4 sockets.

The following methods are available:

- **lpf** (Linux packet filter): This method is available only on Linux and is the default on Linux platforms.
- **bpf** (Berkeley packet filter): This method is available only on FreeBSD and is the default on FreeBSD.
- **socket** (datagram socket API): This method is available on Linux and FreeBSD platforms.

**-s** <server-address>

Specify the server IP address or fully qualified domain name to use as a destination for DHCP protocol messages before **dhclient** has acquired an IP address.

Normally, **dhclient** transmits these messages to 255.255.255.255 (the IP limited broadcast address). Overriding this is mostly useful for debugging purposes. This feature is not supported in DHCPv6 mode (`-6`).

**-g** <relay>

Set the *giaddr* field of all packets to the relay IP address simulating a relay agent. This is for testing purposes only and should not be expected to work in any consistent or useful way.

**-i**

Use a DUID with DHCPv4 clients. If no DUID is available in the lease file, one will be constructed and saved. The DUID will be used to construct an **RFC 4361** style client ID that will be included in the client's messages. This client ID can be overridden by setting a client ID in the configuration file. Overriding the client ID in this fashion is discouraged.

**-I**

Use the standard DNS UPDATE scheme from **RFC 4701** and **RFC 4702**.

**-v**

Print program version and exit.

### 2.2.5.1 Options available for DHCPv6 only

**-S**

Use Information-request to get only stateless configuration parameters (i.e., without address). This implies `-6`. It also doesn't rewrite the lease database.

**-T**

Ask for IPv6 temporary addresses, one set per `-T` argument. This implies `-6` and also disables the normal address query. See `-N` to restore it.

**-P**

Enable IPv6 prefix delegation. This implies `-6` and also disables the normal address query. See `-N` to restore it. Multiple prefixes can be requested with multiple `-P` arguments.

---

**Note:** Only one requested interface is allowed.

---

**-R**

Require that responses include all of the items requested by any `-N`, `-T`, or `-P` options. Normally even if the command line includes a number of these the client will be willing to accept the best lease it can even if the lease doesn't include all of the requested items. This option causes the client to only accept leases that include all of the requested items.

**Warning:** Using this option may prevent the client from using any leases it receives if the servers aren't configured to supply all of the items.

**-D** <LL|LLT>

Override the default when selecting the type of DUID to use. By default, DHCPv6 **dhclient** creates an identifier based on the link-layer address (DUID-LL) if it is running in stateless mode (with **-S**, not requesting an address), or it creates an identifier based on the link-layer address plus a timestamp (DUID-LLT) if it is running in stateful mode (without **-S**, requesting an address). When DHCPv4 is configured to use a DUID using **-i** option the default is to use a DUID-LLT. **-D** overrides these default, with a value of either LL or LLT.

**-N**

Restore normal address query for IPv6. This implies **-6**. It is used to restore normal operation after using **-T** or **-P**. Multiple addresses can be requested with multiple **-N** arguments.

**--dad-wait-time** <seconds>

Specify maximum time (in seconds) that the client should wait for the duplicate address detection (DAD) to complete on an interface. This value is propagated to the *dhclient-script* (8) in a `dad_wait_time` environment variable. If any of the IPv6 addresses on the interface are tentative (DAD is in progress), the script will wait for the specified number of <seconds> for DAD to complete. If the script ignores this variable, then the parameter has no effect.

### 2.2.5.2 Modifying default file locations

The following options may be used to modify the locations a client uses for its files. For example, they can be particularly useful if `/var` or `/var/run` have not been mounted when the **dhclient** process is started.

**-cf** <config-file>

Specify path to the *dhclient.conf* (5) configuration file.

**-df** <duid-lease-file>

Specify path to a secondary lease file. If the primary lease file doesn't contain a DUID, this file will be searched. The DUID read from the secondary will be written to the primary. This option can be used to allow an IPv4 instance of the client to share a DUID with an IPv6 instance. After starting one of the instances the second can be started with this option pointing to the lease file of the first instance. There is no default. If no file is specified no search is made for a DUID should one not be found in the main lease file.

**-lf** <lease-file>

Specify path to the *dhclient.leases* (5) leases file.

**-pf** <pid-file>

Specify path to a file where the **dhclient** process ID is written.

**--no-pid**

Disable writing process ID files. By default, **dhclient** will write a process ID file. If it is invoked with this option it will not check for an existing **dhclient** process.

**-sf** <script-file>

Specify path to the *dhclient-script*(8) network configuration script invoked by **dhclient** when it gets a lease.

## 2.2.6 UDP and TCP ports

During operations the client may use multiple UDP and TCP ports to provide different functions. Which ports are used depends on the configuration in use. The following should provide an idea of what ports may be used.

Normally a DHCPv4 client will open a raw UDP socket to receive and send most DHCPv4 packets. It also opens a fallback UDP socket for use in sending unicast packets. Normally these will both use the well known port number for BOOTPC.

For DHCPv6 the client opens a UDP socket on the well known client port and a fallback UDP socket on a random port for use in sending unicast messages. Unlike DHCPv4 the well known socket doesn't need to be opened in raw mode.

If there is an *omapi-port* statement in the configuration file, then the client will open a TCP socket on that port to listen for OMAPI connections. When something connects another port will be used for the established connection.

When DNS UPDATE is used, the client will open a v4 and a v6 UDP socket on random ports. These ports are not opened unless/until the client first attempts to do an update. If the client is not configured to do DNS UPDATES, the ports will never be opened.

## 2.2.7 Configuration

See *dhclient.conf*(5) for the syntax of **dhclient**'s configuration file.

## 2.2.8 OMAPI

**dhclient** provides the capability to control some aspects of it while it is running, without stopping it. This capability is currently provided using OMAPI (Object Management Application Programming Interface) --- an API for manipulating remote objects. OMAPI clients connect to the **dhclient** process using TCP/IP, authenticate, and can then examine the client's current status and make changes to it.

Rather than implementing the underlying OMAPI protocol directly, user programs should use the *dhcpcctl*(5) API or *omapi*(5) itself. *dhcpcctl*(5) is a wrapper that handles some of the housekeeping chores that OMAPI does not do automatically.

**Warning:** The above paragraph about *dhcpcctl*(5) and *omapi*(5) should be replaced with a discussion of using *omshell*(1).

Most things you'd want to do with the client can be done directly using the **omshell (1)** command, rather than having to write a special program.

**Warning:** This section has to be rewritten, or moved to the developer documentation.

### 2.2.8.1 The control object

The control object allows **dhclient** to be shutdown gracefully, releasing all leases that it holds and deleting any DNS records it may have added. It also allows the DHCP client to be paused --- this unconfigures any interfaces the client is using. It can then be restarted, which causes it to reconfigure those interfaces. The client would typically be paused prior to going into hibernation or sleep on a laptop computer. It would then be resumed after power comes back. This allows PC cards to be shutdown while the computer is hibernating or sleeping, and then reinitialized to their previous state once the computer comes out of hibernation or sleep.

The control object has one attribute --- the state attribute.

- To shutdown the client, its state attribute must be set to 2. It will automatically do a DHCPRELEASE.
- To pause the client, its state attribute must be set to 3.
- To resume the client, its state attribute must be set to 4.

### 2.2.9 Environment variables

The following environment variables may be defined to override the builtin defaults for file locations.

---

**Note:** The use of the corresponding command-line options will ignore the corresponding environment variable settings.

---

#### **PATH\_DHCLIENT\_CONF**

Path to the *dhclient.conf* (5) configuration file.

#### **PATH\_DHCLIENT\_DB**

Path to the *dhclient.leases* (5) database.

#### **PATH\_DHCLIENT\_PID**

Path to the process ID file.

#### **PATH\_DHCLIENT\_SCRIPT**

Path to the *dhclient-script* (8) file.

## 2.2.10 Files

`/usr/sbin/dhclient-script`

The DHCP client network configuration script. See *dhclient-script(8)* for more details.

`/etc/lease/dhclient.conf`

The configuration file for the **dhclient** program. See *dhclient.conf(5)* for more details.

`/var/lib/lease/dhclient.leases`

The leases file. See *dhclient.leases(5)* for more details.

`/var/lib/lease/dhclient.leases~`

Old leases file.

`/var/run/lease/dhclient.pid`

The default process ID file.

## 2.2.11 See also

*dhcpd(8)*, *dhcrelay(8)*, *dhclient-script(8)*, *dhclient.conf(5)*, *dhclient.leases(5)*, *dhcp-eval(5)*, *omshell(1)*

## 2.2.12 Copyright

Copyright (C) 2025-2026 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2004-2017 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1996-2003 by Internet Software Consortium.

## 2.3 dhclient-script --- DHCP client network configuration script

### 2.3.1 Description

The DHCP client network configuration script is invoked from time to time by *dhclient(8)*. This script is used by the DHCP client to set each interface's initial configuration prior to requesting an address, to test the address once it has been offered, and to set the interface's final configuration once a lease has been acquired. If no lease is acquired, the script is used to test predefined leases, if any, and is also called once if no valid lease can be identified.

This script is not meant to be customized by the end user. If local customizations are needed, they should be possible using the enter and exit hooks provided (see the section on [Hooks](#) for details). These hooks will allow the user to override the default behaviour of the client in creating a `/etc/resolv.conf` file.

## 2.3.2 Hooks

When the client script starts, it first defines a shell function, `make_resolv_conf`, which is later used to create the `/etc/resolv.conf` file. To override the default behaviour, redefine this function in the enter hook script.

On after defining the `make_resolv_conf` function, the client script checks for the presence of an executable `/etc/lease/dhclient-enter-hooks` script, and if present, it invokes the script inline, using `bash(1)`'s `.` command. The entire environment documented in the section on [Operation](#) is available to this script, which may modify the environment if needed to change the behaviour of the script. If an error occurs during the execution of the script, it can set the `exit_status` variable to a nonzero value, and **dhclient-script** will exit with that error code immediately after the client script exits.

After all processing has completed, **dhclient-script** checks for the presence of an executable `/etc/lease/dhclient-exit-hooks` script, which if present is invoked using `bash(1)`'s `.` command. The exit status of **dhclient-script** will be passed to **dhclient-exit-hooks** in the `exit_status` shell variable, and will always be 0 if the script succeeded at the task for which it was invoked. The rest of the environment as described previously for **dhclient-enter-hooks** is also present. The **dhclient-exit-hooks** script can modify the value of `exit_status` to change the exit status of **dhclient-script**.

## 2.3.3 Operation

When `dhclient(8)` needs to invoke the client configuration script, it defines a set of variables in the environment, and then invokes **dhclient-script**. In all cases, `$reason` is set to the name of the reason why the script has been invoked. The following reasons are currently defined: MEDIUM, PREINIT, BOUND, RENEW, REBIND, REBOOT, EXPIRE, FAIL, STOP, RELEASE, NBI, and TIMEOUT.

### 2.3.3.1 MEDIUM

The DHCP client is requesting that an interface's media type be set. The interface name is passed in the `$interface`, and the media type is passed in `$medium`.

### 2.3.3.2 PREINIT

The DHCP client is requesting that an interface be configured as required in order to send packets prior to receiving an actual address. For clients which use the BSD socket library, this means configuring the interface with an IP address of 0.0.0.0 and a broadcast address of 255.255.255.255. For other clients, it may be possible to simply configure the interface up without actually giving it an IP address at all. The interface name is passed in `$interface`, and the media type in `$medium`.

If an IP alias has been declared in `dhclient.conf(5)`, its address will be passed in `$alias_ip_address`, and that IP address alias should be deleted from the interface, along with any routes to it.

### 2.3.3.3 BOUND

The DHCP client has done an initial binding to a new address. The new IP address is passed in `$new_ip_address`, and the interface name is passed in `$interface`. The media type is passed in `$medium`. Any options acquired from the server are passed using the option name described in `dhcp-options(5)`, except that dashes (-) are replaced by underscores (\_) in order to make valid shell variables, and the variable names start with `new_`. So, for example, the new subnet mask would be passed in `$new_subnet_mask`. Options from a non-default universe will have the universe name prepended to the option name, for example `$new_dhcp6_server_id`. The options that the client explicitly requested via a PRL or ORO option are passed with the same option name as above but prepended with `requested_` and with a value of 1, for example `requested_subnet_mask=1`. No such variable is defined for options not requested by the client or options that don't require a request option, such as the IP address (`*_ip_address`) or expiration time (`*_expiry`).

Before actually configuring the address, `dhclient-script` should somehow ARP for it and exit with a non-zero status if it receives a reply. In this case, the client will send a DHCPDECLINE message to the server and acquire a different address. This may also be done in the [RENEW](#), [REBIND](#), or [REBOOT](#) states, but is not required, and indeed may not be desirable.

When a binding has been completed, a lot of network parameters are likely to need to be set up. A new `/etc/resolv.conf` file needs to be created, using the values of `$new_domain_name` and `$new_domain_name_servers` (which may list more than one server, separated by spaces). A default route should be set using `$new_routers`, and static routes may need to be set up using `$new_static_routes`.

If an IP address alias has been declared, it must be set up here. The alias IP address will be written as `$alias_ip_address`, and other DHCP options that are set for the alias (e.g., subnet mask) will be passed in variables named as described previously except prefixed with `$alias_` instead of `$new_`. Care should be taken that the alias IP address not be used if it is identical to the bound IP address (`$new_ip_address`), since the other alias parameters may be incorrect in this case.



#### 2.3.3.4 RENEW

When a binding has been renewed, the script is called as in *BOUND*, except that in addition to all the variables whose names are prefixed with *\$new\_* and *\$requested\_*, there is another set of variables whose names are prefixed with *\$old\_*. Persistent settings that may have changed need to be deleted; for example, if a local route to the bound address is being configured, the old local route should be deleted. If the default route has changed, the old default route should be deleted. If the static routes have changed, the old ones should be deleted. Otherwise, processing can be done as with *BOUND*.

#### 2.3.3.5 REBIND

The DHCP client has rebound to a new DHCP server. This can be handled as with *RENEW*, except that if the IP address has changed, the ARP table should be cleared.

#### 2.3.3.6 REBOOT

The DHCP client has successfully reacquired its old address after a reboot. This can be processed as with *BOUND*.

#### 2.3.3.7 EXPIRE

The DHCP client has failed to renew its lease or acquire a new one, and the lease has expired. The IP address must be relinquished, and all related parameters should be deleted, as in *RENEW* and *REBIND*.

#### 2.3.3.8 FAIL

The DHCP client has been unable to contact any DHCP servers, and any leases that have been tested have not proved to be valid. The parameters from the last lease tested should be deconfigured. This can be handled in the same way as *EXPIRE*.

#### 2.3.3.9 STOP

*dhclient(8)* has been informed to shut down gracefully. **dhclient-script** should unconfigure or shutdown the interface as appropriate.

### 2.3.3.10 RELEASE

*dhclient*(8) has been executed using the `-r` command line argument, indicating that the administrator wishes it to release its lease(s). **dhclient-script** should unconfigure or shutdown the interface.

### 2.3.3.11 NBI

NBI stands for No-Broadcast-Interfaces. *dhclient*(8) was unable to find any interfaces upon which it believed it should commence DHCP. What **dhclient-script** should do in this situation is entirely up to the implementor.

### 2.3.3.12 TIMEOUT

The DHCP client has been unable to contact any DHCP servers. However, an old lease has been identified, and its parameters have been passed in as with *BOUND*. The client configuration script should test these parameters and, if it has reason to believe they are valid, should exit with a value of 0. If not, it should exit with a non-zero value.

The usual way to test a lease is to set up the network as with *REBIND* (since this may be called to test more than one lease) and then ping the first router defined in `$routers`. If a response is received, the lease must be valid for the network to which the interface is currently connected. It would be more complete to try to ping all of the routers listed in `$new_routers`, as well as those listed in `$new_static_routes`, but current scripts do not do this.

## 2.3.4 Bugs

If more than one interface is being used, there's no obvious way to avoid clashes between server-supplied configuration parameters; for example, the stock **dhclient-script** rewrites `/etc/resolv.conf`. If more than one interface is being configured, `/etc/resolv.conf` will be repeatedly initialized to the values provided by one server, and then the other. Assuming the information provided by both servers is valid, this shouldn't cause any real problems, but it could be confusing.

## 2.3.5 See also

*dhclient*(8), *dhcpcd*(8), *dhclient.conf*(5), *dhclient.leases*(5)

## 2.3.6 Copyright

Copyright (C) 2025 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2012,2014,2016 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 2009-2010 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 2004-2005 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1996-2003 by Internet Software Consortium.

## 2.4 dhcrelay - DHCP relay agent

### 2.4.1 Synopsis

```
dhcrelay [ -4 ] [ -d ] [ -q ] [ -a ] [ -D ] [ -p <port> ] [ -nio <method> ] [ -c <count> ] [ -A <length> ] [ -pf <pid-file> ] [ --no-pid ] [ -m <append|replace|forward|discard> ] [ -i <interface0> [ ... -i <interfaceN> ] ] [ -iu <interface0> [ ... -iu <interfaceN> ] ] [ -id <interface0> [ ... -id <interfaceN> ] ] [ -U <interface> ] <server0> [ ... <serverN> ]
```

```
dhcrelay -6 [ -d ] [ -q ] [ -I ] [ -p <port> ] [ -c <count> ] [ -pf <pid-file> ] [ --no-pid ] [ -s <subscriber-id> ] [ -l <lower0> [ ... -l <lowerN> ] ] [ -u <upper0> [ ... -u <upperN> ] ]
```

```
dhcrelay [ -h | -V ]
```

### 2.4.2 Description

**dhcrelay** relays DHCP and BOOTP requests, from a subnet to which no DHCP server is directly connected, to one or more DHCP servers on other subnets. It supports both DHCPv4/BOOTP and DHCPv6 protocols.

### 2.4.3 Operation

**dhcrelay** listens for DHCPv4 or DHCPv6 queries from clients or other relay agents on one or more interfaces, passing them along to upstream servers or relay agents as specified on the command line. When a reply is received from upstream, it is multicast or unicast back downstream to the source of the original request.

## 2.4.4 Options

### 2.4.4.1 Protocol selection

**-6**

Run **dhcrelay** as a DHCPv6 relay agent. Incompatible with **-4**.

**-4**

Run **dhcrelay** as a DHCPv4/BOOTP relay agent. This is the default mode of operation, so the argument is not necessary, but may be specified for clarity. Incompatible with **-6**.

### 2.4.4.2 Specifying DHCPv4/BOOTP servers

In DHCPv4 mode, a list of one or more server addresses must be specified on the command line, to which DHCP/BOOTP queries should be relayed.

### 2.4.4.3 Options available for both DHCPv4 and DHCPv6

**-c** *<count>*

Specify the maximum hop count. When forwarding packets, **dhcrelay** discards packets which have reached a hop count of *<count>*. The default value is 10. Maximum value that is allowed is 255.

**-d**

Force **dhcrelay** to run as a foreground process.

**-h**

Print program usage information and exit.

**-p** *<port>*

Listen and transmit on port *<port>*. This is mostly useful for debugging purposes. The default is port 67 for DHCPv4/BOOTP, and port 547 for DHCPv6.

**-nio** *<method>*

Specify the network I/O method to use for DHCP.

By default, **dhcrelay** uses an appropriate default network I/O method suitable for the platform it is built for. This is currently a "raw" method such as LPF on Linux and BPF on BSDs respectively. In this case, it also uses a fallback datagram socket to perform unicast communications when it may need address resolution.

The raw network I/O methods were implemented as it was not previously possible to receive and specify interface information for broadcast datagrams when reading and writing from regular UDP v4 sockets, or even to send datagrams addressed to INADDR\_BROADCAST correctly at all. However, **dhcrelay** is now able to do so on all its supported platforms using regular UDP v4 sockets.

The following methods are available:

- **lpf** (Linux packet filter): This method is available only on Linux and is the default on Linux platforms.
- **bpf** (Berkeley packet filter): This method is available only on FreeBSD and is the default on FreeBSD.
- **socket** (datagram socket API): This method is available on Linux and FreeBSD platforms.

**-q**

Quiet mode. Prevents **dhcrelay** from printing its network configuration on startup.

**-pf** <pid-file>

Specify path to a file where the **dhcrelay** process ID is written.

**--no-pid**

Disable writing process ID files. By default, **dhcrelay** will write a process ID file.

**-v**

Print program version and exit.

#### 2.4.4.4 Options available for DHCPv4 only

**-a**

Append an agent option field to each request before forwarding it to the server. Agent option fields in responses sent from servers to clients will be stripped before forwarding such responses back to the client. The agent option field will contain two agent options: the Circuit ID suboption and the Remote ID suboption. Currently, the Circuit ID will be the printable name of the interface on which the client request was received. The client supports inclusion of a Remote ID suboption as well, but this is not used by default.

**-A** <length>

Specify the maximum packet size to send to a DHCPv4/BOOTP server. This might be done to allow sufficient space for addition of relay agent options while still fitting into the Ethernet MTU size.

**-D**

Drop packets from upstream servers if they contain Relay Agent Information options that indicate they were generated in response to a query that came via a different relay agent. If this option is not specified, such packets will be relayed anyway.

**-i** <ifname>

Listen for DHCPv4/BOOTP traffic on interface <ifname>. Multiple interfaces may be specified by using more than one **-i** option. If no interfaces are specified on the command line, **dhcrelay** will identify all network interfaces, eliminating non-broadcast interfaces if possible, and attempt to listen on all of them.

**-iu** <ifname>

Specifies an upstream network interface --- an interface from which replies from servers and other relay agents will be accepted. Multiple interfaces may be specified by using more than one *-iu* option. This argument is intended to be used in conjunction with one or more *-i* or *-id* arguments.

**-id** <ifname>

Specifies a downstream network interface --- an interface from which requests from clients and other relay agents will be accepted. Multiple interfaces may be specified by using more than one *-id* option. This argument is intended to be used in conjunction with one or more *-i* or *-iu* arguments.

**-m** <append|replace|forward|discard>

Control the handling of incoming DHCPv4 packets which already contain relay agent options. If such a packet does not have *giaddr* set in its header, the DHCP standard requires that the packet be discarded. However, if *giaddr* is set, the relay agent may handle the situation in four ways:

- it may append its own set of relay options to the packet, leaving the supplied option field intact;
- it may replace the existing agent option field;
- it may forward the packet unchanged; or,
- it may discard it.

**-U** <ifname>

Enables the addition of a [RFC 3527](#) compliant link selection suboption for clients directly connected to the relay. This RFC allows a relay to specify two different IP addresses --- one for the server to use when communicating with the relay (*giaddr*), the other for choosing the subnet for the client (the suboption). This can be useful if the server is unable to send packets to the relay via the address used for the subnet.

When enabled, **dhcrelay** will add an agent option (as per *-a* above) that includes the link selection suboption to the forwarded packet. This will only be done to packets received from clients that are directly connected to the relay (i.e. *giaddr* is zero). The address used in the suboption will be that of the link upon which the inbound packet was received (which would otherwise be used for *giaddr*). The value of *giaddr* will be set to that of interface <ifname>.

Only one interface should be marked in this fashion. Currently enabling this option on an interface causes the relay to process all DHCP traffic similar to the *-i* option, in the future we may split the two more completely.

This option is off by default. Note that enabling this option automatically enables the *-a* option.

Using options such as *-m replace* or *-m discard* on relays upstream from one using *-U* can pose problems. The upstream relay will wipe out the initial agent option containing the link selection while leaving the re-purposed *giaddr* value in place, causing packets to go astray.

#### 2.4.4.5 Options available for DHCPv6 only

**-I**

Force use of the DHCPv6 Interface-ID option. This option is automatically sent when there are two or more downstream interfaces in use, to disambiguate between them. The **-I** option causes **dhcrelay** to send the option even if there is only one downstream interface.

**-s** <subscriber-id>

Add an option with the specified <subscriber-id> into the packet. This feature is for testing rather than production as it will put the same <subscriber-id> into the packet for all clients.

**-l** [<address>%]<ifname>[#<index>]

Specifies the *lower* network interface for DHCPv6 relay mode --- the interface on which queries will be received from clients or from other relay agents. At least one **-l** option must be included in the command line when running in DHCPv6 mode. The interface name <ifname> is a mandatory parameter. The link address can be specified by <address>%; if it isn't, **dhcrelay** will use the first non-link-local address configured on the interface. The optional #<index> parameter specifies the interface index.

**-u** [<address>%]<ifname>

Specifies the *upper* network interface for DHCPv6 relay mode --- the interface to which queries from clients and other relay agents should be forwarded. At least one **-u** option must be included in the command line when running in DHCPv6 mode. The interface name <ifname> is a mandatory parameter. The destination unicast or multicast address can be specified by <address>%; if not specified, the relay agent will forward to the DHCPv6 All\_DHCP\_Relay\_Agents\_and\_Servers multicast address.

It is possible to specify the same interface with different addresses more than once, and even, when the system supports it, to use the same interface as both upper and lower interfaces.

#### 2.4.5 See also

*dhclient(8), dhcpcd(8)*

#### 2.4.6 Bugs

Using the same interface on both upper and lower sides may cause loops, so when running this way, the maximum hop count should be set to a low value.

The loopback interface is not (yet) recognized as a valid interface.

## 2.4.7 Copyright

Copyright (C) 2025-2026 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2004-2016 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1997-2003 by Internet Software Consortium.

## 2.5 `omshell` --- OMAPI command shell

### 2.5.1 Synopsis

```
omshell
```

```
omshell [ -h | -V ]
```

### 2.5.2 Description

**omshell** provides an interactive way to connect to, query, and possibly change a DHCP server's state via OMAPI --- the Object Management API. By using OMAPI and **omshell**, changes can be made to the state of a DHCP server while the server is running. **omshell** provides a way of accessing OMAPI.

OMAPI is simply a communications mechanism that allows manipulation of objects. In order to actually use **omshell**, there should be an understanding of what objects are available and how to use them. Documentation for OMAPI objects can be found in the documentation for the server that provides them, in the *dhcpcd(8)* and *dhclient(8)* manual pages.

### 2.5.3 Options

**-h**

Print program usage information and exit.

**-V**

Print program version and exit.

### 2.5.4 Local and remote objects

Throughout this document, there are references to local and remote objects. Local objects are ones created in **omshell** with the `new` command. Remote objects are ones on the DHCP server --- leases, hosts, and groups that the DHCP server knows about. Local and remote objects are associated together to enable viewing and modification of object attributes. Also, new remote objects can be created to match local objects.



## 2.5.5 Opening a connection

**omshell** is started from the command line. Once **omshell** is started, there are several commands that can be executed:

**server** <address>

This command specifies the IP address of the DHCP server to connect to. If it is not specified, the default server is 127.0.0.1 (localhost).

**port** <number>

This command specifies the OMAPI port number of the DHCP server. By default, this is 7911.

**key** <name> <secret>

This command specifies the TSIG key to use to sign OMAPI transactions. <name> is the name of a key defined in *dhcpcd.conf*(5) using the `omapi-key` statement. <secret> is the secret key generated using **dnssec-keygen**(1) (from the Loop DNS distribution), or using another key generation program.

**connect**

This command starts the OMAPI connection to the server provided by the `server` command.

## 2.5.6 Creating local objects

Any object defined in OMAPI can be created, queried, and/or modified. The object types available to OMAPI are defined in *dhcpcd*(8) and *dhclient*(8). When using **omshell**, objects are first defined locally, manipulated as desired, and then associated with an object on the server. Only one object can be manipulated at a time. To create a local object, the following command is used:

**new** <object-type>

<object-type> is one of `group`, `host`, or `lease`.

At this point, a new object would have been created on which properties can be set. For example, if a new lease object was created with new lease, any of a lease's attributes can be set using the following command:

**set** <attribute-name> = <value>

Attribute names are defined in *dhcpcd*(8) and *dhclient*(8). Values should be quoted if they are strings. So, to set a lease's IP address, the following command may be used:

```
> set ip-address 192.168.4.50
```

## 2.5.7 Associating local and remote objects

At this point, the server can be queried for information about this lease, by using the following command:

**open**

Now, the local lease object that was created --- and IP address set for --- is associated with the corresponding lease object on the DHCP server. All of the lease attributes from the DHCP server are now also the attributes on the local object, and will be shown in **omshe11**.

## 2.5.8 Viewing a remote object

To query a lease of address 192.168.4.50 and find out its attributes, after connecting to the server, the following commands may be used:

```
> new lease
```

This creates a new local lease object.

```
> set ip-address = 192.168.4.50
```

This sets the local object's IP address to be 192.168.4.50.

```
> open
```

Now, if a lease with that IP address exists, all the information the DHCP server has about that particular lease is shown. Any data that isn't readily printable text will show up in colon-separated hexadecimal values. In this example, output from the server for the entire transaction might look like this:

```
> new "lease"
obj: lease
> set ip-address = 192.168.4.50
obj: lease
ip-address = c0:a8:04:32
> open
obj: lease
ip-address = c0:a8:04:32
state = 00:00:00:02
dhcp-client-identifier = 01:00:10:a4:b2:36:2c
client-hostname = "wendelina"
subnet = 00:00:00:06
pool = 00:00:00:07
hardware-address = 00:10:a4:b2:36:2c
hardware-type = 00:00:00:01
ends = dc:d9:0d:3b
starts = 5c:9f:04:3b
```

(continues on next page)

(continued from previous page)

```
tstp = 00:00:00:00
tsfp = 00:00:00:00
cltt = 00:00:00:00
```

As can be seen above, the IP address is represented in hexadecimal, as are the starting and ending times of the lease.

## 2.5.9 Modifying a remote object

Attributes of remote objects are updated by using the **set** command as before, and then issuing an **update** command. The **set** command sets the attributes on the current local object, and the **update** command pushes those changes out to the server.

Continuing with the previous example, output from the server may look as follows if the corresponding **set** and **update** commands are executed:

```
> set client-hostname = "something-else"
obj: lease
ip-address = c0:a8:04:32
state = 00:00:00:02
dhcp-client-identifier = 01:00:10:a4:b2:36:2c
client-hostname = "something-else"
subnet = 00:00:00:06
pool = 00:00:00:07
hardware-address = 00:10:a4:b2:36:2c
hardware-type = 00:00:00:01
ends = dc:d9:0d:3b
starts = 5c:9f:04:3b
tstp = 00:00:00:00
tsfp = 00:00:00:00
cltt = 00:00:00:00
> update
obj: lease
ip-address = c0:a8:04:32
state = 00:00:00:02
dhcp-client-identifier = 01:00:10:a4:b2:36:2c
client-hostname = "something-else"
subnet = 00:00:00:06
pool = 00:00:00:07
hardware-address = 00:10:a4:b2:36:2c
hardware-type = 00:00:00:01
ends = dc:d9:0d:3b
starts = 5c:9f:04:3b
tstp = 00:00:00:00
tsfp = 00:00:00:00
cltt = 00:00:00:00
```

## 2.5.10 New remote objects

New remote objects are created much in the same way that existing server objects are modified. A local object is created using the **new** command, attributes are set as desired, and then the remote object is created with the same properties by using the following command:

### create

Now a new object exists on the DHCP server which matches the properties that were set on the local object. Objects created via OMAPI are saved into the *dhcpcd.leases(5)* file.

For example, the following shows how a new host with the IP address of 192.168.4.40 is created:

```
> new host
obj: host
> set name = "some-host"
obj: host
name = "some-host"
> set hardware-address = 00:80:c7:84:b1:94
obj: host
name = "some-host"
hardware-address = 00:80:c7:84:b1:94
> set hardware-type = 1
obj: host
name = "some-host"
hardware-address = 00:80:c7:84:b1:94
hardware-type = 1
> set ip-address = 192.168.4.40
obj: host
name = "some-host"
hardware-address = 00:80:c7:84:b1:94
hardware-type = 1
ip-address = c0:a8:04:28
> create
obj: host
name = "some-host"
hardware-address = 00:80:c7:84:b1:94
hardware-type = 00:00:00:01
ip-address = c0:a8:04:28
```

The *dhcpcd.leases(5)* file would then have an entry like this in it:

```
host some-host {
    dynamic;
    hardware ethernet 00:80:c7:84:b1:94;
    fixed-address 192.168.4.40;
}
```

The `dynamic;` statement is used to indicate that this host entry did not come from

*dhcpcd.conf*(5), but was created dynamically via OMAPI.

### 2.5.11 Resetting attributes

An attribute may be removed from an object by using the **unset** command. Once an attribute has been unset, the **update** command must be used to update the remote object. So, if the host "some-host" from the previous example should not have a static IP address anymore, the commands run in **omshe11** would look like this:

```
obj: host
name = "some-host"
hardware-address = 00:80:c7:84:b1:94
hardware-type = 00:00:00:01
ip-address = c0:a8:04:28
> unset ip-address
obj: host
name = "some-host"
hardware-address = 00:80:c7:84:b1:94
hardware-type = 00:00:00:01
ip-address = <null>
```

**Warning:** Include the result of running the **unset** command as well in the output above.

### 2.5.12 Refreshing objects

A local object may be refreshed with the current remote object's properties using the **refresh** command. This is useful for objects that change periodically, like leases, to see if they have been updated. This isn't particularly useful for hosts.

### 2.5.13 Deleting objects

Any remote object that can be created can also be destroyed. This is done by creating a new local object, setting attributes, associating the local and remote object using the **open** command, and then using the **remove** command. If the host "some-host" from before was created in error, it could be removed using the following command:

```
obj: host
name = "some-host"
hardware-address = 00:80:c7:84:b1:94
hardware-type = 00:00:00:01
ip-address = c0:a8:04:28
> remove
obj: <null>
```

## 2.5.14 Help

The **help** command prints out all of the commands available in **omshe11** with some syntax pointers.

## 2.5.15 See also

*dhcpcctl(3), dhcpcd(8), dhclient(8), dhcpcd.conf(5), dhclient.conf(5)*

## 2.5.16 Copyright

Copyright (C) 2025 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2012,2014 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 2004,2009 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 2001-2003 by Internet Software Consortium.

## CONFIGURATION AND DATA

The following sections document configuration files and data files of some programs that are part of the Lease software distribution.

### 3.1 `dhcpcd.conf` --- DHCP server configuration

#### 3.1.1 Description

`dhcpcd.conf` is the configuration file for `dhcpcd(8)`, the DHCP server daemon.

`dhcpcd.conf` is a free-form ASCII text file. It is parsed by the recursive-descent parser built into `dhcpcd(8)`. The file may contain extra tabs and newlines for formatting purposes. Keywords in the file are case-insensitive. Comments may be placed anywhere within the file, except within quotes. Comments begin with the `#` character and end at the end of the line.

The file essentially consists of a list of statements. Statements fall into two broad categories: parameters and declarations.

Parameter statements specify how to do something (e.g., how long a lease to offer), whether to do something (e.g., should `dhcpcd(8)` provide addresses to unknown clients), or what parameters to provide to the client (e.g., use the gateway 220.177.244.7).

Declarations are used to describe the topology of the network, to describe clients on the network, to provide addresses that can be assigned to clients, or to apply a group of parameters to a group of declarations. In any group of parameters and declarations, all parameters must be specified before any declarations which depend on those parameters may be specified.

Declarations about network topology include the **shared-network** and the **subnet** declarations. If clients on a subnet are to be assigned addresses dynamically, a **range** declaration must appear within the **subnet** declaration. For clients with statically assigned addresses, or for installations where only known clients will be served, each such client must have a **host** declaration. If parameters are to be applied to a group of declarations which are not related strictly on a per-subnet basis, the **group** declaration can be used.

For every subnet which will be served, and for every subnet to which the DHCP server is connected, there must be one **subnet** declaration, which tells `dhcpcd` how to recognize that an address is on that subnet. A subnet declaration is required for each **subnet** even if no addresses will be dynamically allocated on that subnet.

Some installations have physical networks on which more than one IP subnet operates. For example, if there is a site-wide requirement that 8-bit subnet masks be used, but a department with a single physical ethernet network expands to the point where it has more than 254 nodes, it may be necessary to run two 8-bit subnets on the same ethernet until such time as a new physical network can be added. In this case, the **subnet** declarations for these two networks must be enclosed in a **shared-network** declaration.

---

**Note:** Even when the **shared-network** declaration is absent, an empty one is created by the server to contain the **subnet** (and any scoped parameters included in the **subnet**). For practical purposes, this means that "stateless" DHCP clients, which are not tied to addresses (and therefore subnets) will receive the same configuration as stateful ones.

---

Some sites may have departments which have clients on more than one subnet, but it may be desirable to offer those clients a uniform set of parameters which are different than what would be offered to clients from other departments on the same subnet. For clients which will be declared explicitly with **host** declarations, these declarations can be enclosed in a **group** declaration along with the parameters which are common to that department. For clients whose addresses will be dynamically assigned, class declarations and conditional declarations may be used to group parameter assignments based on information the client sends.

When a client is to be booted, its boot parameters are determined by consulting that client's **host** declaration (if any), and then consulting any class declarations matching the client, followed by the **pool**, **subnet**, and **shared-network** declarations for the IP address assigned to the client. Each of these declarations itself appears within a lexical scope, and all declarations at less specific lexical scopes are also consulted for client option declarations. Scopes are never considered twice, and if parameters are declared in more than one scope, the parameter declared in the most specific scope is the one that is used.

When `dhcpcd(8)` tries to find a host declaration for a client, it first looks for a **host** declaration which has a **fixed-address** declaration that lists an IP address that is valid for the subnet or shared network on which the client is booting. If it doesn't find any such entry, it tries to find an entry which has no **fixed-address** declaration.



### 3.1.2 Examples

A typical `dhcpcd.conf` file may look like this:

```
# global parameters...
option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;

subnet 204.254.239.0 netmask 255.255.255.224 {
    # subnet-specific parameters...
    option routers 204.254.239.1;
    range 204.254.239.10 204.254.239.30;
}

subnet 204.254.239.32 netmask 255.255.255.224 {
    # subnet-specific parameters...
    option routers 204.254.239.33;
    range 204.254.239.42 204.254.239.62;
}

subnet 204.254.239.64 netmask 255.255.255.224 {
    # subnet-specific parameters...
    option routers 204.254.239.65;
    range 204.254.239.74 204.254.239.94;
}

group {
    # group-specific parameters...
    host v0.test.example.org {
        # host-specific parameters...
    }
    host v1.test.example.org {
        # host-specific parameters...
    }
    host v2.test.example.org {
        # host-specific parameters...
    }
}
```

Notice that at the beginning of the example, there's a place for global parameters. These might be things like the organization's DNS domain name, the addresses of the DNS nameservers (if they are common to the entire organization), and so on.

As seen in the global parameters in the example above, host addresses in parameters can be specified using their DNS domain names rather than their numeric IP addresses. If a given hostname resolves to more than one IP address (for example, if that host has two ethernet interfaces), then where possible, both addresses are supplied to the client.

Each **subnet** declaration may have subnet-specific parameters. The most obvious reason for having subnet-specific parameters, such as in the example above, is that each

subnet out of necessity has its own router specified using the **router** option.

As seen in the host-specific parameters in the example above, the address is specified numerically. This is not required --- if there is a different domain name for each interface on the router, it's perfectly legitimate to use the domain name for that interface instead of the IP address. However, in many cases there may be only one domain name for all of a router's IP addresses, and it would not be appropriate to use that name here.

There is also a group statement in the example above, which provides common parameters for a set of three hosts --- `v0`, `v1`, and `v2`. These hosts are all in the `test.example.org` domain, so it might make sense for a group-specific parameter to override the domain name supplied to these hosts:

```
option domain-name "test.example.org";
```

Also, given the domain they're in, these are probably test machines. If we wanted to test the DHCP leasing mechanism, we might set the lease timeout somewhat shorter than the default:

```
max-lease-time 120;
default-lease-time 120;
```

You may have noticed that while some parameters start with the **option** keyword, some do not. Parameters starting with the **option** keyword correspond to actual DHCP options, while parameters that do not start with the **option** keyword either control the behavior of the DHCP server (e.g., how long a lease *dhcpcd(8)* will give out), or specify client parameters that are not optional in the DHCP protocol (for example, **server-name** and **filename**).

In the first example above, each host had host-specific parameters. These could include the **hostname** option, the name of a file to upload (the **filename** parameter) and the address of the server from which to upload the file (the **next-server** parameter). In general, any parameter can appear anywhere that parameters are allowed, and will be applied according to the scope in which the parameter appears.

Imagine that you have a site with a lot of NCD X-Terminals. These terminals come in a variety of models, and you want to specify the boot files for each model. One way to do this would be to have host declarations for each server and group them by model:

```
group {
    filename "Xncd19r";
    next-server ncd-booter;

    host ncd1 { hardware ethernet 0:c0:c3:49:2b:57; }
    host ncd4 { hardware ethernet 0:c0:c3:80:fc:32; }
    host ncd8 { hardware ethernet 0:c0:c3:22:46:81; }
}

group {
    filename "Xncd19c";
```

(continues on next page)

(continued from previous page)

```
next-server ncd-booter;

host ncd2 { hardware ethernet 0:c0:c3:88:2d:81; }
host ncd3 { hardware ethernet 0:c0:c3:00:14:11; }
}

group {
  filename "XncdHMX";
  next-server ncd-booter;

  host ncd1 { hardware ethernet 0:c0:c3:11:90:23; }
  host ncd4 { hardware ethernet 0:c0:c3:91:a7:8; }
  host ncd8 { hardware ethernet 0:c0:c3:cc:a:8f; }
}
```

### 3.1.3 Address pools

The **pool** and **pool6** declarations can be used to specify a pool of addresses that will be treated differently than another pool of addresses, even on the same network segment or subnet. For example, you may want to provide a large set of addresses that can be assigned to DHCP clients that are registered to your DHCP server, while providing a smaller set of addresses, possibly with short lease times, that are available for unknown clients. If you have a firewall, you may be able to arrange for addresses from one pool to be allowed access to the internet, while addresses in another pool are not, thus encouraging users to register their DHCP clients. To do this, you would set up a pair of pool declarations:

It is also possible to set up entirely different subnets for known and unknown clients --- address pools exist at the level of shared networks, so address ranges within pool declarations can be on different subnets.

As you can see in the preceding example, pools can have permit lists that control which clients are allowed access to the pool and which aren't. Each entry in a pool's permit list is introduced with the **allow** or **deny** keyword. If a pool has a permit list, then only those clients that match specific entries on the permit list will be eligible to be assigned addresses from the pool. If a pool has a deny list, then only those clients that do not match any entries on the deny list will be eligible. If both permit and deny lists exist for a pool, then only clients that match the permit list and do not match the deny list will be allowed access.

The **pool6** declaration is similar to the **pool** declaration. Currently it is only allowed within a **subnet6** declaration, and may not be included directly in a shared network declaration. In addition to the **range6** statement it allows the **prefix6** statement to be included. You may include **range6** statements for both NA and TA and **prefix6** statements in a single **pool6** statement.

### 3.1.4 Dynamic address allocation

Address allocation is actually only done when a client is in the INIT state and has sent a DHCPDISCOVER message. If the client thinks it has a valid lease and sends a DHCPREQUEST to initiate or renew that lease, the server has only three choices --- it can ignore the DHCPREQUEST, send a DHCPNAK to tell the client it should stop using the address, or send a DHCPACK, telling the client to go ahead and use the address for a while.

If the server finds the address the client is requesting, and that address is available to the client, the server will send a DHCPACK. If the address is no longer available, or the client isn't permitted to have it, the server will send a DHCPNAK. If the server knows nothing about the address, it will remain silent, unless the address is incorrect for the network segment to which the client has been attached and the server is authoritative for that network segment, in which case the server will send a DHCPNAK even though it doesn't know about the address.

There may be a **host** declaration matching the client's identification. If that **host** declaration contains a **fixed-address** declaration that lists an IP address that is valid for the network segment to which the client is connected, the DHCP server will never do dynamic address allocation. In this case, the client is *required* to take the address specified in the **host** declaration. If the client sends a DHCPREQUEST for some other address, the server will respond with a DHCPNAK.

When the DHCP server allocates a new address for a client (remember, this only happens if the client has sent a DHCPDISCOVER), it first looks to see if the client already has a valid lease on an IP address, or if there is an old IP address the client had before that hasn't yet been reassigned. In that case, the server will take that address and check it to see if the client is still permitted to use it. If the client is no longer permitted to use it, the lease is freed if the server thought it was still in use --- the fact that the client has sent a DHCPDISCOVER proves to the server that the client is no longer using the lease.

If no existing lease is found, or if the client is forbidden to receive the existing lease, then the server will look in the list of address pools for the network segment to which the client is attached for a lease that is not in use and that the client is permitted to have. It looks through each pool declaration in sequence (all **range** declarations that appear outside of **pool** declarations are grouped into a single pool with no permit list). If the permit list for the pool allows the client to be allocated an address from that pool, the pool is examined to see if there is an address available. If so, then the client is tentatively assigned that address. Otherwise, the next pool is tested. If no addresses are found that can be assigned to the client, no response is sent to the client.

If an address is found that the client is permitted to have, and that has never been assigned to any client before, the address is immediately allocated to the client. If the address is available for allocation but has been previously assigned to a different client, the server will keep looking in hopes of finding an address that has never before been assigned to a client.

The DHCP server generates the list of available IP addresses from a hash table. This means that the addresses are not sorted in any particular order, and so it is not possible

to predict the order in which the DHCP server will allocate IP addresses.

### 3.1.5 IP address conflict prevention

The DHCP server checks IP addresses to see if they are in use before allocating them to clients. It does this by sending an ICMP Echo request message to the IP address being allocated. If no ICMP Echo reply is received within a second, the address is assumed to be free. This is only done for leases that have been specified in **range** statements, and only when the lease is thought by the DHCP server to be free, i.e., the DHCP server or its failover peer has not listed the lease as in use.

If a response is received to an ICMP Echo request, the DHCP server assumes that there is a configuration error --- the IP address is in use by some host on the network that is not a DHCP client. It marks the address as abandoned, and will not assign it to clients. The lease will remain abandoned for a minimum of **abandon-lease-time** seconds.

If a DHCP client tries to get an IP address, but none are available, but there are abandoned IP addresses, then the DHCP server will attempt to reclaim an abandoned IP address. It marks one IP address as free, and then does the same ICMP Echo request check described previously. If there is no answer to the ICMP Echo request, the address is assigned to the client.

The DHCP server does not cycle through abandoned IP addresses if the first IP address it tries to reclaim is free. Rather, when the next DHCPDISCOVER comes in from the client, it will attempt a new allocation using the same method described here, and will typically try a new IP address.

### 3.1.6 DHCP failover

*dhcpcd*(8) supports the DHCP failover protocol as documented in [draft-ietf-dhc-failover-12](#). This is not not a final protocol document, and we have not done interoperability testing with other vendors' implementations of this protocol, so you must not assume that this implementation conforms to the standard. If you wish to use the DHCP failover feature, ensure that both failover peers are running the same version of *dhcpcd*(8).

The failover protocol allows two DHCP servers (and no more than two) to share a common address pool. Each server will have about half of the available IP addresses in the pool at any given time for allocation. If one server fails, the other server will continue to renew leases out of the pool, and will allocate new addresses out of the roughly half of available addresses that it had when communications with the other server were lost.

It is possible during a prolonged failure, to tell the remaining server that the other server is down, in which case the remaining server will (over time) reclaim all the addresses the other server had available for allocation, and begin to reuse them. This is called putting the server into the PARTNER-DOWN state.

You can put the server into the PARTNER-DOWN state either by using the `omshell(1)` command or by stopping the server, editing the last failover state declaration in the lease file, and restarting the server. If you use this last method, change the **my state** statement's state to `partner-down`:

```
failover peer *<name>* state {  
    my state partner-down;  
    peer state *<state>* at *<date>*;  
}
```

It is only required to change the **my state** statement as shown above.

When the other server comes back online, it should automatically detect that it had been offline, and request a complete update from the server that was running in the PARTNER-DOWN state, and then both servers will resume processing together.

**Warning:** It is possible to get into a dangerous situation: if you put one server into the PARTNER-DOWN state, and then *that* server goes down, and the other server comes back up, the other server will not know that the first server was in the PARTNER-DOWN state, and may issue addresses previously issued by the other server to different clients, resulting in IP address conflicts. Before putting a server into PARTNER-DOWN state, therefore, ensure that the other server will not restart automatically.

The failover protocol defines a primary server role and a secondary server role. There are some differences in how primaries and secondaries act, but most of the differences simply have to do with providing a way for each peer to behave in the opposite way from the other. So one server must be configured as primary, and the other must be configured as secondary, and it doesn't matter too much which one is which.

### 3.1.6.1 Failover startup

When a server starts that has not previously communicated with its failover peer, it must establish communications with its failover peer and synchronize with it before it can serve clients. This can happen either because you have just configured your DHCP servers to perform failover for the first time, or because one of your failover servers has failed catastrophically and lost its database.

The initial recovery process is designed to ensure that when one failover peer loses its database and then resynchronizes, any leases that the failed server gave out before it failed will be honored. When the failed server starts up, it notices that it has no saved failover state, and attempts to contact its peer.

When it has established contact, it asks the peer for a complete copy of its peer's lease database. The peer then sends its complete database, and sends a message indicating that it is done. The failed server then waits until MCLT (the Maximum Client Lead Time) has passed, and once MCLT has passed both servers make the transition back

into normal operation. This waiting period ensures that any leases the failed server may have given out while out of contact with its partner will have expired.

While the failed server is recovering, its partner remains in the PARTNER-DOWN state, which means that it is serving all clients. The failed server provides no service at all to DHCP clients until it has made the transition into normal operation.

In the case where both servers detect that they have never before communicated with their partner, they both come up in this recovery state and follow the procedure we have just described. In this case, no service will be provided to DHCP clients until MCLT has expired.

### 3.1.6.2 Configuring failover

In order to configure failover, you need to write a peer declaration that configures the failover protocol, and you need to write peer references in each **pool** declaration for which you want to do failover. You do not have to do failover for all pools on a given network segment. You must not tell one server it's doing failover on a particular address pool and tell the other it is not. You must not have any common address pools on which you are not doing failover. A **pool** declaration that utilizes failover would look like this:

```
pool {  
    failover peer "foo";  
    pool specific parameters  
};
```

The server currently does very little sanity checking, so if you configure it wrongly, it will just fail in odd ways. It is recommended therefore that you either configure failover or don't configure failover, but don't configure any mixed pools. Also, use the same master configuration file for both servers, and have a separate file that contains the peer declaration and includes the master file. This will help avoid configuration mismatches. As the implementation evolves, this may become less of a problem. A basic sample `dhcpd.conf` file for a primary server might look like this:

```
failover peer "foo" {  
    primary;  
    address a.rc.example.com;  
    port 519;  
    peer address b.rc.example.com;  
    peer port 520;  
    max-response-delay 60;  
    max-unacked-updates 10;  
    mclt 3600;  
    split 128;  
    load balance max seconds 3;  
}  
  
include "/etc/dhcpd.master";
```



The statements in the peer declaration are as follows:

**primary** | **secondary**;

This statement determines whether the server is primary or secondary, as described earlier in the section titled *DHCP failover*.

**address** <address>;

The **address** statement declares the IP address or DNS name on which the server should listen for connections from its failover peer, and also the value to use for the DHCP Failover Protocol server identifier. Because this value is used as an identifier, it may not be omitted.

**peer address** <address>;

The **peer address** statement declares the IP address or DNS name to which the server should connect to reach its failover peer for failover messages.

**port** <port-number>;

The **port** statement declares the TCP port on which the server should listen for connections from its failover peer. This statement may be, in which case the IANA assigned port number 647 will be used by default.

**peer port** <port-number>;

The **peer port** statement declares the TCP port to which the server should connect to reach its failover peer for failover messages. This statement may be omitted, in which case the IANA assigned port number 647 will be used by default.

**max-response-delay** <seconds>;

The **max-response-delay** statement tells the DHCP server how many seconds may pass without receiving a message from its failover peer before it assumes that connection has failed. This number should be small enough that a transient network failure that breaks the connection will not result in the servers being out of communication for a long time, but large enough that the server isn't constantly making and breaking connections. This parameter must be specified.

**max-unacked-updates** <count>;

The **max-unacked-updates** statement tells the remote DHCP server how many BNDUPD messages it can send before it receives a BNDACK from the local system. We don't have enough operational experience to say what a good value for this is, but 10 seems to work. This parameter must be specified.

**mclt** <seconds>;

The **mclt** statement defines the Maximum Client Lead Time. It must be specified on the primary, and may not be specified on the secondary. This is the length of time for which a lease may be renewed by either failover peer without contacting the other. The longer you set this, the longer it



will take for the running server to recover IP addresses after moving into PARTNER-DOWN state. The shorter you set it, the more load your servers will experience when they are not communicating. A value of something like 3600 is probably reasonable, but again bear in mind that we have no real operational experience with this.

**split** <bits>;

The **split** statement specifies the split between the primary and secondary for the purposes of load balancing. Whenever a client makes a DHCP request, the DHCP server runs a hash on the client identification, resulting in value from 0 to 255. This is used as an index into a 256-bit field. If the bit at that index is set, the primary is responsible. If the bit at that index is not set, the secondary is responsible. The split value determines how many of the leading bits are set to one. So, in practice, higher split values will cause the primary to serve more clients than the secondary. Lower split values will cause the secondary to serve more clients than the primary. Legal values are between 0 and 256 inclusive, of which the most reasonable is 128. Note that a value of 0 makes the secondary responsible for all clients and a value of 256 makes the primary responsible for all clients.

---

**Note:** You must only have one of the **split** or **hba** statement defined, never both. For most cases, the fine-grained control that **hba** offers isn't necessary, and **split** should be used.

---

**hba** <colon-separated-hex-list>;

The **hba** statement specifies the split between the primary and secondary as a bitmap rather than a cutoff, which theoretically allows for finer-grained control. In practice, there is probably no need for such fine-grained control.

An example **hba** statement follows:

```
hba ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00;
```

The above is equivalent to a **split 128;** statement, and is identical. The following two examples are also equivalent to a split of 128, but are not identical:

```
hba aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:
    aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa;

hba 55:55:55:55:55:55:55:55:55:55:55:55:55:55:55:
    55:55:55:55:55:55:55:55:55:55:55:55:55:55;
```

They are equivalent, because half the bits are set to 0, half are set to 1 (0xa and 0x5 are 1010 and 0101 in binary respectively) and consequently this would roughly divide the clients equally between the servers. They are not

identical, because the actual peers this would load balance to each server are different for each example.

---

**Note:** You must only have one of the **split** or **hba** statement defined, never both. For most cases, the fine-grained control that **hba** offers isn't necessary, and **split** should be used.

---

**load balance max seconds** *<seconds>;*

This statement allows you to configure a cutoff after which load balancing is disabled. The cutoff is based on the number of seconds since the client sent its first DHCPDISCOVER or DHCPREQUEST message, and only works with clients that correctly implement the *secs* field --- fortunately most clients do. We recommend setting this to something like 3 or 5. The effect of this is that if one of the failover peers gets into a state where it is responding to failover messages but not responding to some client requests, the other failover peer will take over its client load automatically as the clients retry.

**auto-partner-down** *<seconds>;*

This statement instructs the server to initiate a timed delay upon entering the communications-interrupted state (any situation of being out-of-contact with the remote failover peer). At the conclusion of the timer, the server will automatically enter the PARTNER-DOWN state. This permits the server to allocate leases from the partner's free lease pool after an STOS+MCLT timer expires, which can be dangerous if the partner is in fact operating at the time (the two servers will give conflicting bindings).

**Warning:** Think very carefully before enabling this feature. The PARTNER-DOWN and communications-interrupted states are intentionally segregated because there do exist situations where a failover server can fail to communicate with its peer, but still has the ability to receive and reply to requests from DHCP clients. In general, this feature should only be used in those deployments where the failover servers are directly connected to one another, such as by a dedicated hardwired link ("a heartbeat cable").

A *<seconds>* value of 0 disables the **auto-partner-down** feature (which is also the default), and any positive value indicates the time in seconds to wait before automatically entering the PARTNER-DOWN state.

**max-lease-misbalance** *<percentage>;*

**max-lease-ownership** *<percentage>;*

**min-balance** *<seconds>;*

**max-balance** <seconds>;

These are failover pool balance related statements.

*dhcpcd* (8) evaluates pool balance on a schedule, rather than on demand as leases are allocated. The on-demand approach proved to be slightly klunky when pool misbalance reached total saturation --- when any server ran out of leases to assign, it also lost its ability to notice it had run dry.

In order to understand pool balance, some elements of its operation first need to be defined. First, there are *free* and *backup* leases. Both of these are referred to as *free state leases*. *free* and *backup* are the *free states* for the purpose of this document. The difference is that only the primary may allocate from *free* leases unless under special circumstances, and only the secondary may allocate *backup* leases.

When pool balance is performed, the only plausible expectation is to provide a 50/50 split of the free state leases between the two servers. This is because no one can predict which server will fail, regardless of the relative load placed upon the two servers, so giving each server the leases gives both servers the same amount of *failure endurance*. Therefore, there is no way to configure any different behaviour, outside of some very small windows we will describe shortly.

The first thing calculated on any pool balance run is a value referred to as *lts* (Leases To Send). For the primary, this is the difference in the count of free and backup leases, divided by 2. For the secondary, it is the difference in the count of backup and free leases, divided by 2. The resulting value is signed: if it is positive, the local server is to hand out leases to retain a 50/50 balance. If it is negative, the remote server would need to send leases to balance the pool. Once the *lts* value reaches zero, the pool is perfectly balanced (give or take one lease in the case of an odd number of total free state leases).

The current approach is still something of a hybrid of the old approach, marked by the presence of the **max-lease-misbalance** statement. This parameter configures what used to be a 10% fixed value in previous versions: if *lts* is less than: (free + backup \* **max-lease-misbalance**) percent, then the server will skip balancing a given pool --- it won't bother moving any leases, even if some leases should be moved. The meaning of this value is also somewhat overloaded, in that it also governs the estimation of when to attempt to balance the pool --- which may then also be skipped over. The oldest leases in the free and backup states are examined. The time they have resided in their respective queues is used as an estimate to indicate how much time it is probable that it would take before the leases at the top of the list would be consumed, and thus how long it would take to use all leases in that state. This percentage is directly multiplied by this time, and fit into the schedule if it falls within the **min-balance** and **max-balance** configured values. The scheduled pool check time is only moved in a downwards direction, it is never increased. Lastly, if the *lts* is more than double

this number in the negative direction, the local server will *panic* and transmit a failover protocol POOLREQ message, in the hopes that the remote system will be woken up into action.

Once the *lts* value exceeds the **max-lease-misbalance** percentage of total free state leases as described above, leases are moved to the remote server. This is done in two passes.

In the first pass, only leases whose most recent bound client would have been served by the remote server --- according to the Load Balance (see the **split** and **hba** configuration statements) --- are given away to the peer. This first pass will happily continue to give away leases, decrementing the *lts* value by one for each, until the *lts* value has reached the negative of the total number of leases multiplied by the **max-lease-ownership** percentage. So it is through this value that you can permit a small misbalance of the lease pools - for the purpose of giving the peer more than a 50/50 share of leases in the hopes that their clients might some day return and be allocated by the peer (operating normally). This process is referred to as *MAC Address Affinity*, but this is somewhat misnamed: it applies equally to DHCP Client Identifier options. Note also that affinity is applied to leases when they enter the *free* state from the *expired* or *released* states. In this case also, leases will not be moved from free to backup if the secondary already has more its share.

The second pass is only entered into if the first pass fails to reduce the *lts* underneath the total number of free state leases multiplied by **max-lease-ownership** percentage. In this pass, the oldest leases are given over to the peer without second thought about the Load Balance Algorithm, and this continues until the *lts* falls under this value. In this way, the local server will also happily keep a small percentage of leases that would normally load balance to itself.

So, the **max-lease-misbalance** value acts as a behavioural gate. Smaller values will cause more leases to transition states to balance the pools over time, higher values will decrease the amount of change, but may lead to pool starvation if there's a run on leases.

The **max-lease-ownership** value permits a small (percentage) skew in the lease balance of a percentage of the total number of free state leases.

Finally, the **min-balance** and **max-balance** statements make certain that a scheduled rebalance event happens within a reasonable timeframe, for example, not to be thrown off by a 7-year old free lease.

Plausible values for the percentages lie between 0 and 100 (inclusive), but values over 50 are indistinguishable from one another --- once *lts* exceeds 50% of the free state leases, one server must therefore have 100% of the leases in its respective free state. It is recommended to select a **max-lease-ownership** value that is lower than the value selected for the **max-lease-misbalance** value. **max-lease-ownership** defaults to 10, and **max-lease-misbalance** defaults to 15.

Plausible values for the **min-balance** and **max-balance** times also range from 0 to  $(2^{32})-1$  (or the limit of your local `time_t` value), but default to values 60 and 3600 respectively (to place balance events between 1 minute and 1 hour).

### 3.1.7 Client classing

Clients can be separated into classes, and treated differently depending on what class they are in. This separation can be done either with a conditional statement, or with a match statement within the **class** declaration. It is possible to specify a limit on the total number of clients within a particular class or subclass that may hold leases at one time, and it is possible to specify automatic subclassing based on the contents of the client packet.

---

**Note:** Classing support for DHCPv6 clients follows the same rules as for DHCPv4 except that support for billing classes has not been added yet.

---

To add clients to classes based on conditional evaluation, you can specify a **match** expression in the class statement:

```
class "ras-clients" {  
    match if substring (option dhcp-client-identifier, 1, 3) = "RAS";  
}
```

Note that whether you use matching expressions or add statements or both to classify clients, you must always write a **class** declaration for any class that you use. If there will be no **match** statement and no in-scope statements for a class, the declaration should be an empty one:

```
class "ras-clients" {  
}
```

#### 3.1.7.1 Subclasses

In addition to classes, it is possible to declare subclasses. A subclass is a class with the same name as a regular class, but with a specific submatch expression which is hashed for quick matching. This is essentially a speed hack --- the main difference between five classes with **match** expressions and one class with five subclasses is that it will be quicker to find the subclasses. Subclasses work as follows:

```
class "allocation-class-1" {  
    match pick-first-value (option dhcp-client-identifier, hardware);  
}  
  
class "allocation-class-2" {
```

(continues on next page)

(continued from previous page)

```
match pick-first-value (option dhcp-client-identifier, hardware);
}

subclass "allocation-class-1" 1:8:0:2b:4c:39:ad;
subclass "allocation-class-2" 1:8:0:2b:a9:cc:e3;
subclass "allocation-class-1" 1:0:0:c4:aa:29:44;

subnet 10.0.0.0 netmask 255.255.255.0 {
  pool {
    allow members of "allocation-class-1";
    range 10.0.0.11 10.0.0.50;
  }
  pool {
    allow members of "allocation-class-2";
    range 10.0.0.51 10.0.0.100;
  }
}
```

The data following the class name in the **subclass** declaration is a constant value for use in matching the **match** expression for the class. When class matching is done, the server will evaluate the **match** expression and then lookup the result in the hash table. If it finds a match, the client is considered a member of both the class and the subclass.

Subclasses can be declared with or without scope. In the above example, the sole purpose of the subclass is to allow some clients access to one address pool, while other clients are given access to the other pool, so these subclasses are declared without scopes. If part of the purpose of the subclass were to define different parameter values for some clients, you might want to declare some subclasses with scopes.

In the above example, if you had a single client that needed some configuration parameters, while most didn't, you might write the following **subclass** declaration for that client:

```
subclass "allocation-class-2" 1:08:00:2b:a1:11:31 {
  option root-path "storage0:/var/diskless/linux";
  filename "/tftpboot/linux.diskless";
}
```

In this example, we've used subclassing as a way to control address allocation on a per-client basis. However, it's also possible to use subclassing in ways that are not specific to clients --- for example, to use the value of the **vendor-class-identifier** option to determine what values to send in the **vendor-encapsulated-options** option. An example of this is shown in the section on vendor encapsulated options in the *dhcp-options(5)* manual page.

### 3.1.7.2 Per-class limits on dynamic address allocation

You may specify a limit to the number of clients in a class that can be assigned leases. The effect of this will be to make it difficult for a new client in a class to get an address. Once a class with such a limit has reached its limit, the only way a new client in that class can get a lease is for an existing client to relinquish its lease, either by letting it expire, or by sending a DHCPRELEASE packet. Classes with lease limits are specified as follows:

```
class "limited-1" {  
    lease limit 4;  
}
```

This will produce a class in which a maximum of 4 members may hold a lease at one time.

### 3.1.7.3 Spawning classes

It is possible to declare a *spawning class*. A spawning class is a class that automatically produces subclasses based on what the client sends. The reason that spawning classes were created was to make it possible to create lease-limited classes on the fly. The envisioned application is a cable-modem environment where the ISP wishes to provide clients at a particular site with more than one IP address, but does not wish to provide such clients with their own subnet, nor give them an unlimited number of IP addresses from the network segment to which they are connected.

Many cable modem head-end systems can be configured to add a Relay Agent Information option to DHCP packets when relaying them to the DHCP server. These systems typically add a circuit ID or remote ID option that uniquely identifies the customer site. To take advantage of this, you can write a class declaration as follows:

```
class "customer" {  
    spawn with option agent.circuit-id;  
    lease limit 4;  
}
```

Now whenever a request comes in from a customer site, the circuit ID option will be checked against the class's hash table. If a subclass is found that matches the circuit ID, the client will be classified in that subclass and treated accordingly. If no subclass is found matching the circuit ID, a new one will be created and logged in the *dhcpcd.leases(5)* file, and the client will be classified in this new class. Once the client has been classified, it will be treated according to the rules of the class, including, in this case, being subject to the per-site limit of four leases.

The use of the subclass spawning mechanism is not restricted to relay agent options --- this particular example is given only because it is a fairly straightforward one.

### 3.1.7.4 Combining match, match if, and spawn with

In some cases, it may be useful to use one expression to assign a client to a particular class, and a second expression to put it into a subclass of that class. This can be done by combining the **match if** and **spawn with** statements, or the **match if** and **match** statements. For example:

```
class "jr-cable-modems" {
    match if option dhcp-vendor-identifier = "jr-cm";
    spawn with option agent.circuit-id;
    lease limit 4;
}

class "dv-dsl-modems" {
    match if option dhcp-vendor-identifier = "dv-dsl";
    spawn with option agent.circuit-id;
    lease limit 16;
}
```

This allows you to have two classes that both have the same **spawn with** expression without getting the clients in the two classes confused with each other.

### 3.1.8 Dynamic DNS updates

*dhcpcd* (8) has the ability to perform dynamic DNS updates ([RFC 2136](#)). Within the configuration files, you can define how you want the DNS to be updated. These updates are [RFC 2136](#) compliant so any DNS server supporting it should be able to accept updates from the DHCP server.

There are two DNS schemes implemented. The `interim` option is based on draft revisions of the DDNS documents while the `standard` option is based on the RFCs for DHCP-DNS interaction and DHCIDs. The DHCP server may be configured to use one of the methods, or not to do DNS updates.

New installations should use the `standard` option. Older installations may want to continue using the `interim` option for backwards compatibility with the DNS database until the database can be updated. This can be done with the **ddns-update-style** configuration parameter.

### 3.1.9 The DNS UPDATE scheme

The `interim` and `standard` DNS update schemes operate mostly according to specification from the IETF. The `interim` version was based on the drafts in progress at the time while the `standard` is based on the completed RFCs. The standard RFCs are:

- [RFC 4701](#) (updated by [RFC 5494](#))
- [RFC 4702](#)
- [RFC 4703](#)



And the corresponding drafts were:

- draft-ietf-dnsext-dhcid-rr
- draft-ietf-dhc-fqdn-option
- draft-ietf-dhc-ddns-resolution

The basic framework for the two schemes is similar with the main material difference being that a DHCID RR is used in the standard version while the interim versions uses a TXT RR. The format of the TXT record bears a resemblance to the DHCID RR but it is not equivalent (MD5 vs. SHA-256, field length differences, etc.).

In these two schemes, the DHCP server does not necessarily always update both the A and the PTR records. The FQDN option includes a flag which, when sent by the client, indicates that the client wishes to update its own A record. In that case, the server can be configured either to honor the client's intentions or ignore them. This is done with the statements:

**\*\* allow client-updates;** or **\*\* ignore client-updates;**

By default, DNS UPDATES by clients are allowed.

If the server is configured to allow client updates, then if the client sends a fully-qualified domain name in the FQDN option, the server will use that name the client sent in the FQDN option to update the PTR record. For example, let us say that the client is a visitor from the `radish.org` domain, whose hostname is `jschmoe`. The server is for the `example.org` domain. The DHCP client indicates in the FQDN option that its FQDN is `jschmoe.radish.org..` It also indicates that it wants to update its own A record. The DHCP server therefore does not attempt to set up an A record for the client, but does set up a PTR record for the IP address that it assigns the client, with its RDATA pointing to `jschmoe.radish.org`. Once the DHCP client has an IP address, it can update its own A record, assuming that the `radish.org` DNS server will allow it to do so.

If the server is configured not to allow client updates, or if the client doesn't want to do its own update, the server will simply choose a name for the client. By default, the server will choose from the following three values:

- **fqdn** option (if present)
- **hostname** option (if present)
- Configured hostname option (if defined)

If these defaults for choosing the hostname are not appropriate, you can write your own statement to set the **ddns-hostname** variable as you wish. If none of the above are found, the server will use the **host** declaration name (if one exists) and **use-host-decl-names** is true.

It will use its own domain name for the client. It will then update both A and PTR records, using the name that it chose for the client. If the client sends a fully-qualified domain name in the FQDN option, the server uses only the leftmost part of the domain name --- in the example above, `jschmoe` instead of `jschmoe.radish.org..`

Further, if the **ignore client-updates;** directive is used, then the server will in addition send a response in the DHCP packet, using the FQDN option, that implies to the client that it should perform its own updates if it chooses to do so. With **deny client-updates;**, a response is sent which indicates the client may not perform updates.

Both the `standard` and `interim` options also include a method to allow more than one DHCP server to update the DNS without accidentally deleting A records that shouldn't be deleted, or failing to add A records that should be added. For the `standard` option the method works as follows:

When the DHCP server issues a client a new lease, it creates a text string that is a SHA-256 hash over the DHCP client's identification (see [RFC 4701](#) and [RFC 4702](#) for details). The update attempts to add an A record with the name the server chose and a DHCID record containing the hashed identifier string (*hashid*). If this update succeeds, the server is done.

If the update fails because the A record already exists, then the DHCP server attempts to add the A record with the prerequisite that there must be a DHCID record with the same name as the new A record, and that DHCID record's contents must be equal to *hashid*. If this update succeeds, then the client has its A record and PTR record. If it fails, then the name the client has been assigned (or requested) is in use, and can't be used by the client. At this point the DHCP server gives up trying to do a DNS update for the client until the client chooses a new name.

The server also does not update very aggressively. Because each DNS update involves a round trip to the DNS server, there is a cost associated with doing DNS update transactions even if they do not actually modify the DNS database. So the DHCP server tracks whether or not it has updated the record in the past (this information is stored on the lease) and does not attempt to update records that it thinks it has already updated.

This can lead to cases where the DHCP server adds a record, and then the record is deleted through some other mechanism, but the server never again updates the DNS because it thinks the data is already there. In this case the data can be removed from the lease through operator intervention, and once this has been done, the DNS will be updated the next time the client renews.

The `interim` DNS update scheme was written before the RFCs were finalized and does not quite follow them. The RFCs call for a new DHCID RR type while the interim DNS update scheme uses a TXT record. In addition, draft-ietf-dhc-ddns-resolution called for the DHCP server to put a DHCID RR on the PTR record, but the `interim` update method does not do this. In the final RFC this requirement was relaxed such that a server may add a DHCID RR to the PTR record.

### 3.1.9.1 Dynamic DNS UPDATE security

When you set your DNS server up to allow updates from the DHCP server, you may be exposing it to unauthorized updates. To avoid this, you should use TSIG Secret Key Transaction Authentication for DNS ([RFC 8945](#)) --- a method of cryptographically signing updates with HMACs using a shared secret key. As long as you protect the secrecy of this key, your DNS updates should also be secure. Note, however, that the DHCP protocol itself provides no security, and that clients can therefore provide information to the DHCP server which the DHCP server will then use in its DNS updates, with the constraints described previously.

The DNS server must be configured to allow updates for any zone that the DHCP server will be updating. For example, let us say that clients in the `example.org` domain will be assigned addresses on the 10.10.17.0/24 subnet, and that you are using `named(8)` from the [Loop DNS software distribution](#). In that case, in `named.conf(5)`, you will need a **key** declaration for the TSIG key you will be using, and also two **zone** declarations --- one for the zone containing A records that will be updated, and one for the zone containing PTR records:

```
key "DHCP_UPDATER" {
    algorithm hmac-sha256;
    secret "qayUgAHVsBBHwF/lbKRod2jGFDf9c1Oh8az5RYDmFEU=";
};

zone "example.org" {
    type master;
    file "example.org.db";
    allow-update { key DHCP_UPDATER; };
};

zone "17.10.10.in-addr.arpa" {
    type master;
    file "10.10.17.db";
    allow-update { key DHCP_UPDATER; };
};
```

You will also have to configure your DHCP server to do updates to these zones. To do so, you need to add something like this to your `dhcpd.conf` file:

```
key DHCP_UPDATER {
    algorithm hmac-sha256;
    secret "qayUgAHVsBBHwF/lbKRod2jGFDf9c1Oh8az5RYDmFEU=";
};

zone EXAMPLE.ORG. {
    primary 127.0.0.1;
    key DHCP_UPDATER;
}

zone 17.127.10.in-addr.arpa. {
```

(continues on next page)

(continued from previous page)

```
primary 127.0.0.1;
key DHCP_UPDATER;
}
```

The **primary** statement specifies the IP address of the DNS nameserver whose zone information is to be updated. In addition to the **primary** statement, there are also the **primary6**, **secondary**, and **secondary6** statements. The **primary6** statement specifies an IPv6 address for the DNS nameserver. The secondaries provide additional addresses for DNS nameservers to be used if the primary does not respond. The number of DNS nameservers the DDNS code will attempt to use before giving up is limited and is currently set to 3.

Note that the zone declarations have to correspond to authority records in your DNS nameserver --- in the above example, there must be an SOA record for `example.org`. and for `17.10.10.in-addr.arpa..` For example, if there were a subdomain `foo.example.org` with no separate SOA, you could not write a zone declaration for `foo.example.org`. Also keep in mind that DNS zone names in your DHCP configuration should end with a period (`.`); this is the preferred syntax. If you do not end your DNS zone names with a period (`.`), the DHCP server will figure it out. Also note that in the DHCP configuration, zone names are not surrounded by quotes, whereas there are in the DNS configuration.

You should choose your own TSIG secret key, of course. The [Loop DNS software distribution](#) comes with a program for generating TSIG secret keys called *ddns-confgen* (1). Using it, the TSIG secret key can be created as follows:

```
$ ddns-confgen -q -k DHCP_UPDATER
key "DHCP_UPDATER" {
    algorithm hmac-sha256;
    secret "1SBmw/sSPYice0HcTzoJwbt1+6d5sE0V5kfAdgTeD6U=";
};
$
```

The key name, algorithm, and secret must match that being used by the DNS server. The DHCP server currently supports the following TSIG algorithms:

- HMAC-MD5
- HMAC-SHA1
- HMAC-SHA224
- HMAC-SHA256
- HMAC-SHA384
- HMAC-SHA512

You may wish to enable logging of DNS updates on your DNS server. If you are using the [Loop DNS software distribution](#), you can add a **logging** statement like the following to the *named.conf* (5) file:

```
logging {
    channel update_debug {
        file "/var/log/loop/update-debug.log";
        severity debug 3;
        print-category yes;
        print-severity yes;
        print-source yes;
        print-time yes;
    };
    channel security_info {
        file "/var/log/loop/named-auth.info";
        severity info;
        print-category yes;
        print-severity yes;
        print-source yes;
        print-time yes;
    };

    category update { update_debug; };
    category security { security_info; };
};
```

You must create the `/var/log/loop/named-auth.info` and `/var/log/loop/update-debug.log` files before running the `named(8)` program. For more information, see `named.conf(5)`.

### 3.1.10 Events

There are three kinds of events that can happen regarding a lease, and it is possible to declare statements that occur when any of these events happen. These events are the commit event when the server has made a commitment of a certain lease to a client, the release event when the client has released the server from its commitment, and the expiry event when the commitment expires.

To declare a set of statements to execute when an event happens, you must use the **on** statement, followed by the name of the event, followed by a series of statements to execute when the event happens, enclosed in braces.

### 3.1.11 Declarations

**include** <filename-string>;

The **include** statement is used to read in the specified file, and process the contents of that file as though it were entered in the lexical place of the **include** statement.

**shared-network** <name> { [ <parameters> ] [ <declarations> ] }

The **shared-network** statement is used to inform the DHCP server that some IP subnets actually share the same physical network. Any subnets in a shared network should be declared within a **shared-network** statement. Parameters specified in the **shared-network** statement will be used when booting clients on those subnets unless parameters provided at the subnet or host level override them. If any subnet in a shared network has addresses available for dynamic allocation, those addresses are collected into a common pool for that shared network and assigned to clients as needed. There is no way to distinguish on which subnet of a shared network a client should boot.

*<name>* should be the name of the shared network. This name is used when printing debugging messages, so it should be descriptive for the shared network. The name may have the syntax of a valid domain name (although it will never be used as such), or it may be any arbitrary name, enclosed in quotes.

**subnet** *<subnet-number>* **netmask** *<netmask>* { [ *<parameters>* ] [ *<declarations>* ] }

The **subnet** statement is used to provide *dhcpcd*(8) with enough information to tell whether or not an IP address is on that subnet. It may also be used provide subnet-specific parameters and to specify which addresses may be dynamically allocated to clients booting on that subnet. Such addresses are specified using the **range** declaration.

The *<subnet-number>* should be an IP address or domain name which resolves to the subnet number of the subnet being described. The *<netmask>* should be an IP address or domain name which resolves to the subnet mask of the subnet being described. The subnet number, together with the netmask, are sufficient to determine whether any given IP address is on the specified subnet.

Although a netmask must be given with every subnet declaration, it is recommended that if there is any variance in subnet masks at a site, a **subnet-mask** option statement be used in each subnet declaration to set the desired subnet mask, since any **subnet-mask** option statement will override the subnet mask declared in the **subnet** statement.

**subnet6** *<subnet6-number>* { [ *<parameters>* ] [ *<declarations>* ] }

The **subnet6** statement is used to provide *dhcpcd*(8) with enough information to tell whether or not an IPv6 address is on that subnet6. It may also be used to provide subnet-specific parameters and to specify what addresses may be dynamically allocated to clients booting on that subnet.

The *<subnet6-number>* should be an IPv6 network identifier, specified as ip6-address/bits.

**range** [ **dynamic-bootp** ] *<low-address>* [ *<high-address>* ] ;

For any subnet on which addresses will be assigned dynamically, there must be at least one **range** statement. The **range** statement provides the lowest and highest IP addresses in a range. All IP addresses in the range

should be in the subnet in which the **range** statement is declared. The **dynamic-bootp** flag may be specified if addresses in the specified range may be dynamically assigned to BOOTP clients as well as DHCP clients. When specifying a single address, high-address can be omitted.

```
range6 <low-address> <high-address>;  
range6 <subnet6-number>;  
range6 <subnet6-number> temporary;  
range6 <address> temporary;
```

For any IPv6 subnet6 on which addresses will be assigned dynamically, there must be at least one **range6** statement. The **range6** statement can either be the lowest and highest IPv6 addresses in a range6, or use CIDR notation specified as ip6-address/bits. All IP addresses in the range6 should be in the subnet6 in which the **range6** statement is declared.

The **temporary** flag if set makes the prefix (by default on 64 bits) available for temporary addresses ([RFC 4941](#)). A new address per prefix in the shared network is computed at each request with an IA\_TA option. Release and Confirm ignores temporary addresses.

Any IPv6 addresses given to hosts with **fixed-address6** are excluded from the range6, as are IPv6 addresses on the server itself.

```
prefix6 <low-address> <high-address> / <bits>;
```

The **prefix6** statement is the **range6** equivalent for Prefix Delegation ([RFC 3633](#)). Prefixes of <bits> length are assigned between <low-address> and <high-address>.

Any IPv6 prefixes given to static entries (hosts) with **fixed-prefix6** are excluded from the prefix6.

This statement is currently global but it should have a shared-network scope.

```
host <hostname> { [ <parameters> ] [ <declarations> ] }
```

The **host** declaration provides a way for the DHCP server to identify a DHCP or BOOTP client. This allows the server to provide configuration information including fixed addresses or, in DHCPv6, fixed prefixes for specific clients.

If it is desirable to be able to boot a DHCP or BOOTP client on more than one subnet with fixed v4 addresses, more than one address may be specified in the **fixed-address** declaration, or more than one **host** statement may be specified matching the same client.

The **fixed-address6** declaration is used for v6 addresses. At this time it only works with a single address. For multiple addresses specify multiple **host**

statements.

If client-specific boot parameters must change based on the network to which the client is attached, then multiple **host** declarations should be used. The **host** declarations will only match a client if one of their **fixed-address** statements is viable on the subnet (or shared network) where the client is attached. Conversely, for a **host** declaration to match a client being allocated a dynamic address, it must not have any **fixed-address** statements. You may therefore need a mixture of **host** declarations for any given client --- some having **fixed-address** statements, others without.

*<hostname>* should be a name identifying the host. If a *hostname* option is not specified for the host, *<hostname>* is used.

**host** declarations are matched to actual DHCP or BOOTP clients by matching the **dhcp-client-identifier** option specified in the **host** declaration to the one supplied by the client, or, if the **host** declaration or the client does not provide a **dhcp-client-identifier** option, by matching the **hardware** parameter in the **host** declaration to the network hardware address supplied by the client. BOOTP clients do not normally provide a **dhcp-client-identifier**, so the hardware address must be used for all clients that may boot using the BOOTP protocol.

DHCPv6 servers can use the **host-identifier** option parameter in the **host** declaration, and specify any option with a fixed value to identify hosts.

Please be aware that only the **dhcp-client-identifier** option and the hardware address can be used to match a **host** declaration, or the **host-identifier** option parameter for DHCPv6 servers. For example, it is not possible to match a **host** declaration to a **host-name** option. This is because the **host-name** option cannot be guaranteed to be unique to any given client, whereas both the hardware address and **dhcp-client-identifier** option are at least theoretically guaranteed to be unique to a given client.

**group** { [ *<parameters>* ] [ *<declarations>* ] }

The **group** statement is used to apply one or more parameters to a group of declarations. It can be used to group hosts, shared networks, subnets, or even other groups.

### 3.1.12 Allow and deny

The **allow** and **deny** statements can be used to control the response of the DHCP server to various sorts of requests. The **allow** and **deny** keywords actually have different meanings depending on the context. In a pool context, these keywords can be used to set up access control lists for address allocation pools. In other contexts, the keywords simply control general server behavior with respect to clients based on scope. In a non-pool context, the **ignore** keyword can be used in place of the **deny** keyword to prevent logging of denied requests.



### 3.1.12.1 allow, deny, and ignore in scope

The following usages of **allow** and **deny** will work in any scope, although it is not recommended that they be used in pool declarations.

**allow unknown-clients;**  
**deny unknown-clients;**  
**ignore unknown-clients;**

The **unknown-clients** flag is used to tell *dhcpcd(8)* whether or not to dynamically assign addresses to unknown clients. Dynamic address assignment to unknown clients is allowed by default. An unknown client is simply a client that has no host declaration.

The use of this option is now deprecated. If you are trying to restrict access on your network to known clients, you should use **deny unknown-clients;** inside of your address pool, as described in the section titled *allow and deny within pool declarations*.

**allow bootp;**  
**deny bootp;**  
**ignore bootp;**

The **bootp** flag is used to tell *dhcpcd(8)* whether or not to respond to BOOTP queries. BOOTP queries are allowed by default.

**allow booting;**  
**deny booting;**  
**ignore booting;**

The **booting** flag is used to tell *dhcpcd(8)* whether or not to respond to queries from a particular client. This keyword only has meaning when it appears in a **host** declaration. By default, booting is allowed, but if it is disabled for a particular client, then that client will not be able to get an address from the DHCP server.

**allow duplicates;**

**deny duplicates;**

**host** declarations can match client messages based on the DHCP Client Identifier option or based on the client's network hardware type and MAC address. If the MAC address is used, the **host** declaration will match any client with that MAC address --- even clients with different client identifiers. This doesn't normally happen, but is possible when one computer has more than one operating system installed on it --- for example, Microsoft Windows and NetBSD or Linux.

The **duplicates** flag tells the DHCP server that if a request is received from a client that matches the MAC address of a host declaration, any other leases matching that MAC address should be discarded by the server, even if the UID is not the same. This is a violation of the DHCP protocol, but can prevent clients whose client identifiers change regularly from holding many leases at the same time. By default, duplicates are allowed.

**allow declines;**

**deny declines;**

**ignore declines;**

The DHCPDECLINE message is used by DHCP clients to indicate that the lease the server has offered is not valid. When the server receives a DHCPDECLINE for a particular address, it normally abandons that address, assuming that some unauthorized system is using it. Unfortunately, a malicious or buggy client can, using DHCPDECLINE messages, completely exhaust the DHCP server's allocation pool. The server will eventually reclaim these leases, but not while the client is running through the pool. This may cause serious thrashing in the DNS, and it will also cause the DHCP server to forget old DHCP client address allocations.

The **declines** flag tells the DHCP server whether or not to honor DHCPDECLINE messages. If it is set to deny or ignore in a particular scope, the DHCP server will not respond to DHCPDECLINE messages.

The **declines** flag is only supported by DHCPv4 servers. Given the large IPv6 address space and the internal limits imposed by the server's address generation mechanism, we don't think it is necessary for DHCPv6 servers at this time.

Currently, abandoned IPv6 addresses are reclaimed in one of two ways:

- Client renews a specific address: If a client using a given DUID submits a DHCP REQUEST containing the last address abandoned by that DUID, the address will be reassigned to that client.
- Upon the second restart following an address abandonment: When an address is abandoned, it is both recorded as such in the lease file and

retained as abandoned in server memory until the server is restarted. Upon restart, the server will process the lease file and all addresses whose last known state is abandoned will be retained as such in memory but not rewritten to the lease file. This means that a subsequent restart of the server will not see the abandoned addresses in the lease file, and will therefore have no record of them as abandoned in memory, and as such, the server will perceive them as free for assignment.

The total number of addresses in a pool, available for a given DUID value, is internally limited by the server's address generation mechanism. If through mistaken configuration, multiple clients are using the same DUID, they will be competing for the same addresses causing the server to reach this internal limit rather quickly. The internal limit isolates this type of activity such that the address range is not exhausted for other DUID values. The appearance of the following error log message can be an indication of this condition:

"Best match for DUID <XX> is an abandoned address, This may be a result of multiple clients attempting to use this DUID"

where <XX> is an actual DUID value depicted as colon separated string of bytes in hexadecimal values.

**allow client-updates;**  
**deny client-updates;**

The **client-updates** flag tells the DHCP server whether or not to honor the client's intention to do its own update of its A record. See the section titled *Dynamic DNS updates* for details.

**allow leasequery;**  
**deny leasequery;**

The **leasequery** flag tells the DHCP server whether or not to answer DHCPLEASEQUERY packets. The answer to a DHCPLEASEQUERY packet includes information about a specific lease, such as when it was issued and when it will expire. By default, the server will not respond to these packets.

### 3.1.12.2 allow and deny within pool declarations

The uses of the **allow** and **deny** keywords shown in the previous section work pretty much the same way whether the client is sending a DHCPDISCOVER or a DHCPREQUEST message --- an address will be allocated to the client (either the old address it's requesting, or a new address) and then that address will be tested to see if it's okay to let the client have it. If the client requested it, and it's not okay, the server will send a DHCPNAK message. Otherwise, the server will simply not respond to the client. If it is okay to give the address to the client, the server will send a DHCPACK message.

The primary motivation behind **pool** declarations is to have address allocation pools whose allocation policies are different. A client may be denied access to one pool, but allowed access to another pool on the same network segment. In order for this to work, access control has to be done during address allocation, not after address allocation is done.

When a DHCPREQUEST message is processed, address allocation simply consists of looking up the address the client is requesting and checking if it's still available for the client. If it is, then the DHCP server checks both the address pool permit lists and the relevant in-scope **allow** and **deny** statements to see if it's okay to give the lease to the client. In the case of a DHCPDISCOVER message, the allocation process is done as described in the section titled *Dynamic address allocation*.

When declaring permit lists for address allocation pools, the following statements are recognized:

**allow known-clients;**  
**deny known-clients;**

If specified, this statement either allows or prevents allocation from this pool to any client that has a host declaration (i.e., is known). A client is known if it has a host declaration in any scope, not just the current scope.

**allow unknown-clients;**  
**deny unknown-clients;**

If specified, this statement either allows or prevents allocation from this pool to any client that has no host declaration (i.e., is not known).

**allow members of <class>;**  
**deny members of <class>;**

If specified, this statement either allows or prevents allocation from this pool to any client that is a member of the named class.

**allow dynamic bootp clients;**  
**deny dynamic bootp clients;**

If specified, this statement either allows or prevents allocation from this pool to any BOOTP client.

**allow authenticated clients;**  
**deny authenticated clients;**

If specified, this statement either allows or prevents allocation from this pool to any client that has been authenticated using the DHCP authentication protocol.

<b>Warning:</b> This is not yet supported.
--

**allow unauthenticated clients;**  
**deny unauthenticated clients;**

If specified, this statement either allows or prevents allocation from this pool to any client that has not been authenticated using the DHCP authentication protocol.

<b>Warning:</b> This is not yet supported.
--

**allow all clients;**  
**deny all clients;**

If specified, this statement either allows or prevents allocation from this pool to all clients. This can be used when you want to write a pool declaration for some reason, but hold it in reserve, or when you want to renumber your network quickly, and thus want the server to force all clients that have

been allocated addresses from this pool to obtain new addresses immediately when they next renew.

**allow after** *<time>*;

**deny after** *<time>*;

If specified, this statement either allows or prevents allocation from this pool after a given date. This can be used when you want to move clients from one pool to another. The server adjusts the regular lease time so that the latest expiry time is at (*<time>* + **min-lease-time**). A short **min-lease-time** enforces a step change, whereas a longer **min-lease-time** allows for a gradual change. *<time>* is either seconds since epoch, or a UTC time string e.g. 4 2007/08/24 09:14:32 or a string with time zone offset in seconds e.g. 4 2007/08/24 11:14:32 -7200.

### 3.1.13 Parameters

**adandon-lease-time** *<time>*;

*<time>* should be the maximum amount of time (in seconds) that an abandoned IPv4 lease remains unavailable for assignment to a client. Abandoned leases will only be offered to clients if there are no free leases. If not defined, the default abandon lease time is 86400 seconds (24 hours). Note that the abandoned lease time for a given lease is preserved across server restarts. The parameter may only be set at the global scope and is evaluated only once during server startup.

Values less than 60 seconds are not recommended as this is below the ping check threshold and can cause leases, once abandoned but since returned to the free state, to not be pinged before being offered. If the requested time is larger than  $0x7FFFFFFF - 1$ , or the sum of the current time plus the abandoned time is greater than  $0x7FFFFFFF$ , it is treated as infinite.

**adaptive-lease-time-threshold** *<percentage>*;

When the number of allocated leases within a pool rises above *<percentage>*, the DHCP server decreases the lease length for new clients within this pool to **min-lease-time** seconds. Clients renewing an already valid (long) leases get at least the remaining time from the current lease. Since the leases expire faster, the server may either recover more quickly or avoid pool exhaustion entirely. Once the number of allocated leases drop below the threshold, the server reverts back to normal lease times. Valid percentages are between 1 and 99.

**always-broadcast** *<flag>*;

The DHCP and BOOTP protocols both require DHCP and BOOTP clients to set the broadcast bit in the flags field of the BOOTP message header. Unfortunately, some DHCP and BOOTP clients do not do this, and therefore may not receive responses from the DHCP server. The DHCP server can be made to always broadcast its responses to clients by setting this flag to true for the relevant scope; relevant scopes would be inside a conditional statement, as a parameter for a class, or as a parameter for a **host** declaration. To avoid creating excess broadcast traffic on your network, we recommend that you restrict the use of this option to as few clients as possible. For example, the Microsoft DHCP client and *dhclient(8)* are known not to have this problem.

**always-reply-rfc1048** <flag>;

Some BOOTP clients expect **RFC 1048**-style responses, but do not follow **RFC 1048** when sending their requests. You can tell that a client is having this problem if it is not getting the options you have configured for it, and if you see in the server log the message "(non-rfc1048)" printed with each BOOTREQUEST that is logged.

If you want to send **RFC 1048** options to such a client, you can set the **always-reply-rfc1048** option in that client's host declaration, and the DHCP server will respond with an **RFC 1048**-style vendor options field. This flag can be set in any scope, and will affect all clients covered that scope.

**authoritative;**

**not authoritative;**

The DHCP server will normally assume that the configuration information about a given network segment is not known to be correct and is not authoritative. This is so that if a naive user installs a DHCP server not fully understanding how to configure it, it does not send spurious DHCPNAK messages to clients that have obtained addresses from a legitimate DHCP server on the network.

Network administrators setting up authoritative DHCP servers for their networks should always write the **authoritative;** statement at the top of their configuration file to indicate that the DHCP server should send DHCPNAK messages to misconfigured clients. If this is not done, clients will be unable to get a correct IP address after changing subnets until their old lease has expired, which could take quite a long time.

Usually, writing **authoritative;** at the top-level of the configuration file should be sufficient. However, if a DHCP server is to be set up so that it is aware of some networks for which it is authoritative and some networks for which it is not, it may be more appropriate to declare authority on a per-network-segment basis.

Note that the most specific scope for which the concept of authority makes any sense is the physical network segment --- either a **shared-network** statement or a **subnet** statement that is not contained within a **shared-network** statement. It is neither meaningful to specify that the server is authoritative for some subnets within a shared network but not authoritative for others, nor is it meaningful to specify that the server is authoritative for some **host** declarations and not others.

**boot-unknown-clients** *<flag>*;

If the **boot-unknown-clients** statement is present and *<flag>* has a value of `false` or `off`, then clients for which there is no **host** declaration will not be allowed to obtain IP addresses. If this statement is not present or *<flag>* has a value of `true` or `on`, then clients without **host** declarations will be allowed to obtain IP addresses as long as those addresses are not restricted by **allow** and **deny** statements within their **pool** declarations. The default is `true`.

**db-time-format** (*default* | *local*);

The DHCP server software outputs several timestamps when writing leases to persistent storage. This configuration parameter selects one of two output formats. The `default` format prints the day, date, and time in UTC, whereas the `local` format prints the system seconds-since-epoch and helpfully provides the day and time in the system timezone in a comment. The time formats are described in detail in *dhcpcd.leases(5)*. The default is `default`.

**ddns-hostname** *<name>*;

The *<name>* parameter should be the hostname that will be used in setting up the client's A and PTR records. If no **ddns-hostname** parameter is specified in scope, then the server will derive the hostname automatically, using an algorithm that varies for each of the different DNS update methods.

**ddns-domainname** *<name>*;

The *<name>* parameter should be the domain name that will be appended to the client's hostname to form a fully-qualified domain-name (FQDN).

**ddns-local-address4** *<address>*;

**ddns-local-address6** *<address>*;

The *<address>* parameter should be the local IPv4 or IPv6 address the server should use as the source address when sending DDNS update requests.

**ddns-rev-domainname** *<name>*;

The *<name>* parameter should be the domain name that will be appended to the client's reversed IP address to produce a name for use in the client's



PTR record. By default, this is "in-addr.arpa.", but the default can be overridden here.

The reversed IP address to which this domain name is appended is always the IP address of the client in dotted quad notation in reversed order --- for example, if the IP address assigned to the client is 10.17.92.74, then the reversed IP address is 74.92.17.10. So a client with that IP address would, by default, be given a PTR record of 10.17.92.74.in-addr.arpa.. See section 3.5 of [RFC 1035](#) for more information.

**ddns-update-style** (*standard* | *interim* | *none*);

The style parameter must be one of *standard*, *interim* or *none*. The **ddns-update-style** statement is only meaningful in the outer scope --- it is evaluated once after reading the *dhcpcd.conf(5)* file, rather than each time a client is assigned an IP address, so there is no way to use different DNS update styles for different clients. The default is *none*. See the section titled *Dynamic DNS updates* for details about the DDNS update style.

**ddns-updates** <flag>;

The **ddns-updates** parameter controls whether or not the server will attempt to do a DNS update when a lease is confirmed. Set this to false if server should not attempt to do DNS updates within a certain scope. The **ddns-updates** parameter is true by default. To disable DNS updates in all scopes, it is preferable to use the **ddns-update-style** statement setting the style to *none*.

**default-lease-time** <time>;

<time> should be the length in seconds that will be assigned to a lease if the client requesting the lease does not ask for a specific expiration time. This is used for both DHCPv4 and DHCPv6 leases. It is also known as the "valid lifetime" in DHCPv6. The default is 43200 seconds.

**delayed-ack** <count>;

**max-ack-delay** <microseconds>;

<count> should be an integer value between 0 and  $2^{16}-1$ , and defaults to 28. The count represents the maximum number of DHCPv4 replies that will be queued pending transmission until after a database commit event. If this number is reached, a database commit event (commonly resulting in *fsync(2)* and representing a performance penalty) will be made, and the reply packets will be transmitted in a batch afterwards. This preserves the [RFC 2131](#) direction that "stable storage" be updated prior to replying to clients. Should the DHCPv4 sockets "go dry" (i.e., *select(2)* or equivalent function returns immediately with no read sockets), a commit is made

and any queued packets are transmitted. If *<count>* is set to 0, the delayed ack feature is turned off and no responses are queued.

Similarly, *<microseconds>* indicates how many microseconds are permitted to pass between queuing a packet pending *fsync(2)*, and performing the *fsync(2)*. Valid values range from 0 to  $2^{32}-1$ , and it defaults to 250000 (1/4 of a second).

---

**Note:** DHCPv4o6 responses cannot be delayed in the current implementation, and will be sent immediately.

---

### **dhcp-cache-threshold** *<percentage>;*

The **dhcp-cache-threshold** statement takes one integer parameter with allowed values between 0 and 100. The default value is 25 (25% of the lease time). This parameter expresses the percentage of the total lease time, measured from the beginning, during which a client's attempt to renew its lease will result in getting the already assigned lease, rather than an extended lease.

Clients that attempt renewal frequently can cause the server to update and write the database frequently resulting in a performance impact on server. The **dhcp-cache-threshold** statement instructs the DHCP server to avoid updating leases too frequently thus avoiding this behavior. Instead, the server assigns the same lease (i.e. reuses it) with no modifications except for CLTT (Client Last Transmission Time) which does not require disk operations. This feature applies to IPv4 only.

When an existing lease is matched to a renewing client, it will be reused if all of the following conditions are true:

1. The **dhcp-cache-threshold** is larger than zero.
2. The current lease is active.
3. The percentage of the lease time that has elapsed is less than **dhcp-cache-threshold**.
4. The client information provided in the renewal does not alter any of the following:
  - DNS information and DNS updates are enabled
  - Billing class to which the lease is associated
  - The host declaration associated with the lease
  - The client ID --- this may happen if a client boots without a client ID and then starts using one in subsequent requests.

Note that the lease can be reused if the options the client or relay agent sends are changed. These changes will not be recorded in the in-memory

or on-disk databases until the client renews after the threshold time is reached.

**do-forward-updates** <flag>;

The **do-forward-updates** statement instructs the DHCP server as to whether it should attempt to update a DHCP client's A record when the client acquires or renews a lease. This statement has no effect unless DNS updates are enabled. Forward updates are enabled by default. If this statement is used to disable forward updates, the DHCP server will never attempt to update the client's A record, and will only ever attempt to update the client's PTR record if the client supplies an FQDN that should be placed in the PTR record using the `fqdn` option. If forward updates are enabled, the DHCP server will still honor the setting of the **client-updates** flag.

**dont-use-fsync** <flag>;

The **dont-use-fsync** statement instructs the DHCP server whether it should call `fsync(2)` when writing leases to the lease file. By default, and if <flag> is set to false, the server will call `fsync(2)`. Suppressing the call to `fsync(2)` may increase the performance of the server but it also adds a risk that a lease will not be properly written to the disk after it has been issued to a client and before the server stops. This can lead to duplicate leases being issued to different clients.

<b>Warning:</b> Using this option is not recommended.
---

**dynamic-bootp-lease-cutoff** <date>;

The *dynamic-bootp-lease-cutoff* statement sets the ending time for all leases assigned dynamically to BOOTP clients. Because BOOTP clients do not have any way of renewing leases, and don't know that their leases could expire, by default `dhcpcd(8)` assigns infinite leases to all BOOTP clients. However, it may make sense in some situations to set a cutoff date for all BOOTP leases --- for example, the end of a school term, or the time at night when a facility is closed and all machines are required to be powered off.

<date> should be the date on which all assigned BOOTP leases will end. The date is specified in the form:

<W> <YYYY>/<MM>/<DD> <HH>:<mm>:<SS>

<W> is the day of the week expressed as a number from zero (Sunday) to six (Saturday). <YYYY> is the year, including the century. <MM> is the month expressed as a number from 1 to 12. <DD> is the day of the month, counting from 1. <HH> is the hour, from zero to 23. <mm> is the minute and <SS> is the second. The time is always in Coordinated Universal Time (UTC), not local time.

**dynamic-bootp-lease-length** <length>;

The **dynamic-bootp-lease-length** statement is used to set the length of leases dynamically assigned to BOOTP clients. At some sites, it may be possible to assume that a lease is no longer in use if its holder has not used BOOTP or DHCP to get its address within a certain time period. The period is specified in *<length>* as a number of seconds. If a client reboots using BOOTP during the timeout period, the lease duration is reset to *<length>*, so a BOOTP client that boots frequently enough will never lose its lease. Needless to say, this parameter should be adjusted with extreme caution.

**echo-client-id** *<flag>*;

The **echo-client-id** statement is used to enable or disable [RFC 6842](#) compliant behavior. If the **echo-client-id** statement is present and has a value of `true` or `on`, and a DHCP DISCOVER or REQUEST is received which contains the client identifier option (Option code 61), the server will copy the option into its response (DHCP ACK or NAK) per [RFC 6842](#). In other words, if the client sends the option it will receive it back. By default, this flag is `off` and client identifiers will not echoed back to the client.

**filename** *<filename>*;

The **filename** statement can be used to specify the name of the initial boot file which is to be loaded by a client. The *<filename>* should be a filename recognizable to whatever file transfer protocol the client can be expected to use to load the file.

**fixed-address** *<address>* [, *<address>* ... ];

The **fixed-address** declaration is used to assign one or more fixed IP addresses to a client. It should only appear in a **host** declaration. If more than one address is supplied, then when the client boots, it will be assigned the address that corresponds to the network on which it is booting. If none of the addresses in the **fixed-address** statement are valid for the network to which the client is connected, that client will not match the **host** declaration containing that **fixed-address** declaration. Each address in the **fixed-address** declaration should be either an IP address or a domain name that resolves to one or more IP addresses.

**fixed-address6** *<ip6-address>*;

The **fixed-address6** declaration is used to assign a fixed IPv6 addresses to a client. It should only appear in a **host** declaration.

**fixed-prefix6** *<low-address>* / *<bits>*;

The **fixed-prefix6** declaration is used to assign a fixed IPv6 prefix to a client. It should only appear in a **host** declaration, but multiple **fixed-prefix6** statements may appear in a single **host** declaration.

The *<low-address>* specifies the start of the prefix, and *<bits>* specifies the size of the prefix in bits.

If there are multiple prefixes for a given host entry the server will choose one that matches the requested prefix size or, if none match, the first one.

If there are multiple **host** declarations the server will try to choose a declaration where the **fixed-address6** matches the client's subnet. If none match it will choose one that doesn't have a **fixed-address6** statement.

---

**Note:** Unlike the fixed address the fixed prefix does not need to match a subnet in order to be served. This allows you to provide a prefix to a client that is outside of the subnet on which the client makes the request to the server.

---

**get-lease-hostnames** *<flag>*;

The **get-lease-hostnames** statement is used to tell *dhcpcd(8)* whether or not to look up the domain name corresponding to the IP address of each address in the lease pool and use that address for the DHCP hostname option. If *<flag>* is true, then this lookup is done for all addresses in the current scope. By default, or if *<flag>* is false, no lookups are done.

**hardware** *<hardware-type>* *<hardware-address>*;

In order for a BOOTP client to be recognized, its network hardware address must be declared using a **hardware** clause in the **host** statement. *<hardware-type>* must be the name of a physical hardware interface type. Currently, only the **ethernet** type is recognized. The *<hardware-address>* should be a set of hexadecimal octets (numbers from 0 through ff) separated by colons. The **hardware** statement may also be used for DHCP clients.

**host-identifier option** *<option-name>* *<option-data>*;

**host-identifier v6relpt** *<number>* *<option-name>* *<option-data>*;

This identifies a DHCPv6 client in a **host** statement. *<option-name>* is any option, and *<option-data>* is the value for the option that the client will send. *<option-data>* must be a constant value. In the **v6relpt** case the additional *<number>* field is the relay to examine for the specified option name and value. The values are the same as for the **v6relay** option 0 is a no-op, 1 is the relay closest to the client, 2 the next one in, and so on. Values that are larger than the maximum number of relays (currently 32) indicate the relay closest to the server independent of number.

**ignore-client-uids** *<flag>*;

If the **ignore-client-uids** statement is present and has a value of **true** or **on**, the UID for clients will not be recorded. If this statement is not present or has a value of **false** or **off**, then client UIDs will be recorded.

**infinite-is-reserved** *<flag>*;

*dhcpcd(8)* supports *reserved* leases. See the section titled *Reserved leases* for more details. If this *<flag>* is true, the server will automatically reserve

leases allocated to clients which requested an infinite (0xffffffff) lease-time. The default is false.

**lease-file-name** <filename>;

<filename> should be the name of the DHCP server's lease file. By default, this is `/var/lib/lease/dhcpd.leases`. This statement must appear in the outer scope of the configuration file --- if it appears in some other scope, it will have no effect. Furthermore, it has no effect if overridden by `dhcpd(8)`'s `-lf` command line argument, or the `PATH_DHCPD_DB` environment variable.

**limit-addrs-per-ia** <number>;

By default, the DHCPv6 server will limit clients to one IAADDR per IA option, meaning one address. If you wish to permit clients to hang onto multiple addresses at a time, configure a larger number here.

Note that there is no present method to configure the server to forcibly configure the client with one IP address per each subnet on a shared network. This is left to future work.

**dhcpv6-lease-file-name** <filename>;

<filename> is the name of the lease file to use if and only if the server is running in DHCPv6 mode. By default, this is `/var/lib/lease/dhcpd6.leases`. This statement, like **lease-file-name**, must appear in the outer scope of the configuration file. It has no effect if overridden by `dhcpd(8)`'s `-lf` command line argument, or the `PATH_DHCPD6_DB` environment variable. If **dhcpv6-lease-file-name** is not specified, but **lease-file-name** is, the latter's value will be used.

**lease-id-format** (octal | hex);

This parameter governs the format used to write certain values to lease files. With the default format `octal`, values are written as quoted strings in which non-printable characters are represented as octal escapes --- a backslash character followed by three octal digits. When the `hex` format is specified, values are written as an unquoted series of pairs of hexadecimal digits, separated by colons.

Currently, the values written out based on **lease-id-format** are the server-uid, the uid (DHCPv4 leases), and the IAID\_DUID (DHCPv6 leases). Note the server automatically reads the values in either format.

**local-port** <port>;

This statement causes the DHCP server to listen for DHCP requests on the UDP port specified in `port`, rather than on port 67.

**local-address** <address>;

This statement causes the DHCP server to listen for DHCP requests sent to the specified <address>, rather than requests sent to all addresses. Since

serving directly attached DHCP clients implies that the server must respond to requests sent to the broadcast IP address, this option cannot be used if clients are on directly attached networks; it is only realistically useful for a server whose only clients are reached via unicasts, such as via DHCP relay agents.

**Error:** Check if this feature is enabled in the Lease builds. This statement is only effective if the server was compiled using the `USE_SOCKETS #define` statement, which is default on a small number of operating systems, and must be explicitly chosen at compile-time for all others. You can be sure if your server is compiled with `USE_SOCKETS` if you see lines of this format at startup:

Listening on Socket/eth0

---

**Note:** Since this `bind(2)`s all DHCP sockets to the specified address, only one address may be supported in a daemon at a given time.

---

### **log-facility** <facility>;

This statement causes the DHCP server to do all of its logging on the specified log <facility> once the `dhcpcd.conf` file has been read. By default, the DHCP server logs to the daemon facility. Possible log facilities include `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `mark`, `news`, `ntp`, `security`, `syslog`, `user`, `uucp`, and `local0` through `local7`. Not all of these facilities are available on all systems, and there may be other facilities available on other systems. See `syslog(3)`.

In addition to setting this value, you may have to edit your syslog daemon's configuration file to configure logging of the DHCP server. For example, you may have to add a line like this to `syslog.conf`:

```
local7.debug /var/log/dhcpcd.log
```

The syntax of your syslog daemon may be different based on what implementation is being used on your machine; please consult the syslog daemon's documentation. To get the syslog daemon to start logging to the new file, you may first have to create the log file with correct ownership and permissions (e.g., the same ownership and permissions of your `/var/log/messages` file) and restart the syslog daemon. Some systems support log rotation using a program such as `logrotate(8)`, and you may want to configure that as well so that your log file doesn't grow uncontrollably.

Because the **log-facility** setting is configured in the `dhcpcd.conf` file, log messages printed while parsing `dhcpcd.conf` or before parsing it are logged to the default log facility. When this setting is configured, the DHCP server prints its startup message a second time after parsing the configura-

tion file, so that the log will be as complete as possible.

**log-threshold-high** *<percentage>*;

**log-threshold-low** *<percentage>*;

The **log-threshold-low** and **log-threshold-high** statements are used to control when a message is output about pool usage. The value for both of them is the percentage of the pool in use. If the high threshold is 0 or has not been specified, no messages will be produced. If a high threshold is given, a message is output once the pool usage passes that level. After that, no more messages will be output until the pool usage falls below the low threshold. If the low threshold is not given, it defaults to a value of 0.

A special case occurs when the low threshold is set to be higher than the high threshold. In this case, a message will be generated each time a lease is acknowledged when the pool usage is above the high threshold.

Note that threshold logging will be automatically disabled for shared subnets whose total number of addresses is larger than  $(2^{64})-1$ . The server will emit a log statement at startup when threshold logging is disabled as shown below:

"Threshold logging disabled for shared subnet of ranges: *<addresses>*"

This is likely to have no practical runtime effect as CPUs are unlikely to support a server actually reaching such a large number of leases.

**max-lease-time** *<time>*;

*<time>* should be the maximum length in seconds that will be assigned to a lease. If not defined, the default maximum lease time is 86400. The only exception to this is that Dynamic BOOTP lease lengths, which are not specified by the client, are not limited by this maximum.

**min-lease-time** *<time>*;

*<time>* should be the minimum length in seconds that will be assigned to a lease. The default is the minimum of 300 seconds or **max-lease-time**.

**min-secs** *<seconds>*;

*<seconds>* should be the minimum number of seconds since a client began trying to acquire a new lease before the DHCP server will respond to its request. The number of seconds is based on what the client reports, and the maximum value that the client can report is 255 seconds. Generally, setting this to one will result in the DHCP server not responding to the client's first request, but always responding to its second request.



This can be used to set up a secondary DHCP server which never offers an address to a client until the primary server has been given a chance to do so. If the primary server is down, the client will bind to the secondary server, but otherwise clients should always bind to the primary. Note that this does not, by itself, permit a primary server and a secondary server to share a pool of dynamically-allocatable addresses.

**next-server** <server-name>;

The **next-server** statement is used to specify the host address of the server from which the initial boot file (specified in the **filename** statement) is to be loaded. <server-name> should be a numeric IP address or a domain name.

**omapi-port** <port>;

The **omapi-port** statement causes the DHCP server to listen for OMAPI connections on the specified <port>. This statement is required to enable the OMAPI protocol, which is used to examine and modify the state of the DHCP server as it is running.

**one-lease-per-client** <flag>;

If this flag is enabled, whenever a client sends a DHCPREQUEST for a particular lease, the server will automatically free any other leases the client holds. This presumes that when the client sends a DHCPREQUEST, it has forgotten any lease not mentioned in the DHCPREQUEST, i.e., the client has only a single network interface and it does not remember leases it's holding on networks to which it is not currently attached.

**Warning:** Neither of these assumptions are guaranteed or provable, so we urge caution in the use of this statement.

**pid-file-name** <filename>;

<filename> should be the name of the DHCP server's process ID file. This is the file in which the DHCP server's process ID is stored when the server starts. By default, this is /var/run/lease/dhcpd.pid. Like the **lease-file-name** statement, this statement must appear in the outer scope of the configuration file. It has no effect if overridden by *dhcpd(8)*'s `-pf` command line argument, or the `PATH_DHCPD_PID` environment variable.

**dhcpv6-pid-file-name** <filename>;

<filename> should be the name of the DHCP server's process ID file to use if and only if the server is running in DHCPv6 mode. Like the **pid-file-name** statement, this statement must appear in the outer scope of the configuration file. It has no effect if overridden by *dhcpd(8)*'s `-pf` command line argument, or the `PATH_DHCPD6_PID` environment variable. If **dhcpv6-pid-file-name** is not specified, but **pid-file-name** is, the latter's value will be used.

**ping-check** <flag>;

When the DHCP server is considering dynamically allocating an IP address to a client, it first sends an ICMP Echo request (a ping) to the address being assigned. It waits for a second, and if no ICMP Echo response has been heard, it assigns the address. If a response is heard, the lease is abandoned, and the server does not respond to the client. The lease will remain abandoned for a minimum of **abandon-lease-time** seconds.

If there are no free addresses but there are abandoned IP addresses, the DHCP server will attempt to reclaim an abandoned IP address regardless of the value of **abandon-lease-time**.

This ping check introduces a default 1 second delay in responding to DHCPDISCOVER messages, which can be a problem for some clients. The delay may be configured using the **ping-timeout** parameter. The **ping-check** configuration parameter can be used to control checking --- if its value is false, no ping check is done.

**ping-timeout** <seconds>;

If the DHCP server determined it should send an ICMP Echo request (a ping) because the **ping-check** statement is true, **ping-timeout** allows you to configure how many seconds the DHCP server should wait for an ICMP Echo response to be heard. If no ICMP Echo response has been received before the timeout expires, it assigns the address. If a response is heard, the lease is abandoned, and the server does not respond to the client. If it is not configured, **ping-timeout** defaults to 1 second.

**preferred-lifetime** <seconds>;

IPv6 addresses have *valid* and *preferred* lifetimes. The valid lifetime determines at what point at lease might be said to have expired, and is no longer useable. A preferred lifetime is an advisory condition to help applications move off of the address and onto currently valid addresses (should there still be any open TCP sockets or similar).

The preferred lifetime defaults to 5/8 of the **default-lease-time**.

**prefix-length-mode** (*ignore* | *prefer* | *exact* | *minimum* | *maximum*);

According to [RFC 3633](#), DHCPv6 clients may specify preferences when soliciting prefixes by including an IA\_PD Prefix option within the IA\_PD option. Among the preferences that may be conveyed is the prefix-length. When non-zero it indicates a client's desired length for offered prefixes. The RFC states that servers "MAY choose to use the information ... to select prefix(es)" but does not specify any particular rules for doing so. The **prefix-length-mode** statement can be used to set the prefix selection rules employed by the server, when clients send a non-zero prefix-length value. The mode parameter must be one of *ignore*, *prefer*, *exact*, *minimum*, or *maximum* which are described below:

1. *ignore* --- The requested length is ignored. The server will offer the first available prefix.

2. `prefer` --- The server will offer the first available prefix with the same length as the requested length. If none are found then it will the first available prefix of any length.
3. `exact` --- The server will offer the first available prefix with the same length as the requested length. If none are found, it will return a status indicating no prefixes available. This is the default behavior.
4. `minimum` - The server will offer the first available prefix with the same length as the requested length. If none are found, it will return the first available prefix whose length is greater than (e.g. longer than), the requested value. If none of those are found, it will return a status indicating no prefixes available. For example, if client requests a length of /60, and the server has available prefixes of lengths /56 and /64, it will offer prefix of length /64.
5. `maximum` - The server will offer the first available prefix with the same length as the requested length. If none are found, it will return the first available prefix whose length is less than (e.g. shorter than), the requested value. If none of those are found, it will return a status indicating no prefixes available. For example, if client requests a length of /60, and the server has available prefixes of lengths /56 and /64, it will offer a prefix of length /56.

In general "first available" is determined by the order in which pools are defined in the server's configuration. For example, if a subnet is defined with three prefix pools A,B, and C:

```
subnet 3000::/64 {  
    # pool A  
    pool6 {  
        :  
    }  
    # pool B  
    pool6 {  
        :  
    }  
    # pool C  
    pool6 {  
        :  
    }  
}
```

then the pools will be checked in the order A, B, C. For modes `prefer`, `minimum`, and `maximum` this may mean checking the pools in that order twice. A first pass through is made looking for an available prefix of exactly the preferred length. If none are found, then a second pass is performed starting with pool A but with appropriately adjusted length criteria.

The default is mode `exact`.

**remote-port** <port>;

This statement causes the DHCP server to transmit DHCP responses to DHCP clients upon the UDP port specified in *<port>*, rather than on port 68. In the event that the UDP response is transmitted to a DHCP Relay, the server generally uses the local-port configuration value. Should the DHCP Relay happen to be addressed as 127.0.0.1, however, the DHCP Server transmits its response to the remote-port configuration value.

---

**Note:** This is generally only useful for testing purposes, and this configuration value should generally not be used.

---

**server-identifier** *<hostname>*;

The **server-identifier** statement can be used to define the value that is sent in the DHCP Server Identifier option for a given scope. The value specified must be an IP address for the DHCP server, and must be reachable by all clients served by a particular scope.

The use of the **server-identifier** statement is not recommended --- the only reason to use it is to force a value other than the default value to be sent on occasions where the default value would be incorrect. The default value is the first IP address associated with the physical network interface on which the request arrived.

The usual case where the **server-identifier** statement needs to be sent is when a physical interface has more than one IP address, and the one being sent by default isn't appropriate for some or all clients served by that interface. Another common case is when an IP address alias is defined for the purpose of having a consistent IP address for the DHCP server, and it is desired that the clients use this IP address when contacting the server.

Supplying a value for the **dhcp-server-identifier** option is equivalent to using the **server-identifier** statement.

**server-id-check** *<flag>*;

The **server-id-check** statement is used to control whether or not a server, participating in failover, verifies that the value of the **dhcp-server-identifier** option in received DHCP REQUESTs match the server's ID before processing the request. Server ID checking is disabled by default. Setting this flag enables ID checking and thereafter the server will only process requests that match.

---

**Note:** The flag setting should be consistent between failover partners.

---

Unless overridden by use of the **server-identifier** statement, the value the server uses as its ID will be the first IP address associated with the physical network interface on which the request arrived.

In order to reduce runtime overhead the server only checks for a server

ID option in the global and subnet scopes. Complicated configurations may result in different server IDs for this check and when the server ID for a reply packet is determined, which would prohibit the server from responding.

The primary use for this option is when a client broadcasts a request but requires that the response come from a specific failover peer. An example of this would be when a client reboots while its lease is still active --- in this case both servers will normally respond. Most of the time the client won't check the server ID and can use either of the responses. However if the client does check the server ID it may reject the response if it came from the wrong peer. If the timing is such that the "wrong" peer responds first most of the time the client may not get an address for some time.

---

**Note:** Care should be taken before enabling this option.

---

**server-duid** LLT [ *<hardware-type>* *<timestamp>* *<hardware-address>* ];

**server-duid** EN *<enterprise-number>* *<enterprise-identifier>*;

**server-duid** LL [ *<hardware-type>* *<hardware-address>* ];

The **server-duid** statement configures the server DUID. You may pick either **LLT** (link local address plus time), **EN** (enterprise), or **LL** (link local).

If you choose **LLT** or **LL**, you may specify the exact contents of the DUID. Otherwise the server will generate a DUID of the specified type.

If you choose **EN**, you must include the *<enterprise-number>* and the *<enterprise-identifier>*.

If there is a **server-duid** statement in the lease file, it will take precedence over the **server-duid** statement from the config file and a **dhcp6.server-id** option in the config file will override both.

The default **server-duid** type is **LLT**.

**server-name** *<name>*;

The **server-name** statement can be used to inform the client of the name of the server from which it is booting. *<name>* should be the name that will be provided to the client.

**dhcpv6-set-tee-times** *<flag>*;

The **dhcpv6-set-tee-times** statement enables setting T1 and T2 to the values recommended in [RFC 3315](#) section 22.4. When setting T1 and T2, the server will use **dhcp-renewal-time** and **dhcp-rebinding-time** respectively. A value of 0 tells the client it may choose its own value.

When those options are not defined then values will be set to 0 unless the global **dhcpcv6-set-tee-times** parameter is enabled. When this option is enabled, the times are calculated as recommended by [RFC 3315](#) section 22.4:

T1 will be set to 0.5 times the shortest preferred lifetime in the reply. If the "shortest" preferred lifetime is 0xFFFFFFFF, T1 will set to 0xFFFFFFFF.

T2 will be set to 0.8 times the shortest preferred lifetime in the reply. If the "shortest" preferred lifetime is 0xFFFFFFFF, T2 will set to 0xFFFFFFFF.

Keep in mind that given sufficiently small lease lifetimes, the above calculations will result in the two values being equal. For example, a 9 second lease lifetime would yield  $T1 = T2 = 4$  seconds, which would cause clients to issue rebinds only. In such a case it would likely be better to explicitly define the values.

**Warning:** **dhcpcv6-set-tee-times** is intended to be transitional and will likely be removed in a future release. Once removed the behavior will be to use the configured values when present or calculate them per the RFC. If you want zeros, define them as zeros.

#### **site-option-space** <name>;

The **site-option-space** statement can be used to determine from what option space site-local options will be taken. This can be used in much the same way as the **vendor-option-space** statement. Site-local options in DHCP are those options whose numeric codes are greater than 224. These options are intended for site-specific uses, but are frequently used by vendors of embedded hardware that contains DHCP clients. Because site-specific options are allocated on an ad-hoc basis, it is quite possible that one vendor's DHCP client might use the same option code that another vendor's client uses, for different purposes. The **site-option-space** option can be used to assign a different set of site-specific options for each such vendor, using conditional evaluation (see *dhcpcv6-eval* (5) for details).

#### **stash-agent-options** <flag>;

If the **stash-agent-options** parameter is true for a given client, the server will record the relay agent information options sent during the client's initial DHCPREQUEST message when the client was in the SELECTING state and behave as if those options are included in all subsequent DHCPREQUEST messages sent in the RENEWING state. This works around a problem with relay agent information options, which is that they usually do not appear in DHCPREQUEST messages sent by the client in the RENEWING state, because such messages are unicast directly to the server, and are not sent through a relay agent.

#### **update-conflict-detection** <flag>;

If the **update-conflict-detection** parameter is true, the server will perform standard DHCPID multiple-client, one-name conflict detection. If the pa-

parameter has been set false, the server will skip this check and instead simply tear down any previous bindings to install the new binding without question. The default is true.

#### **update-optimization** <flag>;

If the **update-optimization** parameter is false for a given client, the server will attempt a DNS update for that client each time the client renews its lease, rather than only attempting an update when it appears to be necessary. This will allow the DNS to heal from database inconsistencies more easily, but the cost is that the DHCP server must do many more DNS updates. We recommend leaving this option enabled, which is the default. If this parameter is not specified, or is true, the DHCP server will only update when the client information changes, when the client gets a different lease, or when the client's lease expires.

#### **update-static-leases** <flag>;

The **update-static-leases** flag, if enabled, causes the DHCP server to do DNS updates for clients even if those clients are being assigned their IP address using a **fixed-address** statement --- that is, the client is being given a static assignment. It is not recommended because the DHCP server has no way to tell that the DNS update has been done, and therefore will not delete the record when it is not in use. Also, the server must attempt the DNS update each time the client renews its lease, which could have a significant performance impact in environments that place heavy demands on the DHCP server.

#### **use-eui-64** <flag>;

**Warning:** Support for this parameter must be enabled at compile time; see EUI\_64 in `includes/site.h`.

The **use-eui-64** flag, if enabled, instructs the server to construct an address using the client's EUI-64 DUID (Type 3, HW Type EUI-64), rather than creating an address using the dynamic algorithm. This means that a given DUID will always generate the same address for a given pool and further that the address is guaranteed to be unique to that DUID. The IPv6 address will be calculated from the EUI-64 link layer address, conforming to [RFC 2373](#), unless there is a **host** declaration for the client-id.

The **range6** statement for EUI-64 must define full /64 bit ranges. Invalid ranges will be flagged during configuration parsing as errors. See the following example:

```
subnet6 fc00:e4::/64 {
    use-eui-64 true;
    range6 fc00:e4::/64;
}
```

The statement may be specified down to the pool level, allowing a mixture of dynamic and EUI-64 based pools.

During lease file parsing, any leases which map to an EUI-64 pool, that have a non-EUI-64 DUID or for which the lease address is not the EUI-64 address for that DUID in that pool, will be discarded.

If a **host** declaration exists for the DUID, the server grants the address (**fixed-prefix6**, **fixed-address6**) according to the **host** declaration, regardless of the DUID type of the client (even for EUI-64 DUIDs).

If a client requests an EUI-64 lease for a given network, and the resultant address conflicts with a fixed address reservation, the server will send the client a "no addresses available" response.

Any client with a non-conforming DUID (not type 3 or not hw type EUI-64) that is not linked to a host declaration, which requests an address from an EUI-64 enabled pool will be ignored and the event will be logged.

Pools that are configured for EUI-64 will be skipped for dynamic allocation. If there are no pools in the shared network from which to allocate, the client will get back a no addresses available status.

On an EUI-64 enabled pool, any client with a DUID 3, HW Type EUI-64, requesting a solicit/renew and including IA\_NA that do not match the EUI-64 policy, they will be treated as though they are "outside" the subnet for a given client message:

- Solicit - Server will advertise with EUI-64 ia suboption, but with rapid commit off
- Request - Server will send "an address not on link status", and no ia suboption
- Renew/Rebind - Server will send the requested address ia suboption with lifetimes of 0, plus an EUI-64 ia

**use-host-decl-names** *<flag>*;

If the **use-host-decl-names** parameter is true in a given scope, then for every **host** declaration within that scope, the name provided for the **host** declaration will be supplied to the client as its hostname. So, for example:

```
group {
    use-host-decl-names on;

    host joe {
        hardware ethernet 08:00:2b:4c:29:32;
        fixed-address joe.example.com;
    }
}
```

is equivalent to



```
host joe {
    hardware ethernet 08:00:2b:4c:29:32;
    fixed-address joe.example.com;
    option host-name "joe";
}
```

Additionally, enabling **use-host-decl-names** instructs the server to use the **host** declaration name in the forward DNS name, if no other values are available. This value selection process is discussed in more detail in the section titled *Dynamic DNS updates*.

An **option host-name statement** within a host declaration will override the use of the name in the host declaration.

It should be noted here that most DHCP clients completely ignore the host-name option sent by the DHCP server, and there is no way to configure them not to do this. So you generally have a choice of either not having any hostname to client IP address mapping that the client will recognize, or doing DNS updates. It is beyond the scope of this document to describe how to make this determination.

**use-lease-addr-for-default-route** *<flag>;*

If the **use-lease-addr-for-default-route** parameter is true in a given scope, then instead of sending the value specified in the **routers** option (or sending no value at all), the IP address of the lease being assigned is sent to the client. This supposedly causes Win95 machines to ARP for all IP addresses, which can be helpful if your router is configured for proxy ARP.

**Warning:** The use of this feature is not recommended, because it won't work for many DHCP clients.

**vendor-option-space** *<string>;*

The **vendor-option-space** parameter determines from what option space vendor options are taken. The use of this configuration parameter is illustrated in the *dhcp-options(5)* manual page in the section on vendor encapsulated options.

### 3.1.14 Setting parameter values using expressions

Sometimes it's helpful to be able to set the value of a DHCP server parameter based on some value that the client has sent. To do this, you can use expression evaluation. *1dhcp-eval(5)* describes how to write expressions. To assign the result of an evaluation to an option, define the option as follows:

*<my-parameter> = <expression>;*

For example:

```
ddns-hostname = binary-to-ascii (16, 8, "-",  
                                substring (hardware, 1, 6));
```

### 3.1.15 Reserved leases

It's often useful to allocate a single address to a single client, in approximate perpetuity. Host statements with **fixed-address** clauses exist to a certain extent to serve this purpose, but because **host** statements are intended to approximate *static configuration*, they suffer from not being referenced in a littany of other server services, such as dynamic DNS, failover, **on events** and so forth.

If a standard dynamic lease, as from any **range** statement, is marked *reserved*, then the server will only allocate this lease to the client it is identified by (be that by client identifier or hardware address).

In practice, this means that the lease follows the normal state engine, enters ACTIVE state when the client is bound to it, expires, or is released, and any events or services that would normally be supplied during these events are processed normally, as with any other dynamic lease. The only difference is that failover servers treat reserved leases as special when they enter the FREE or BACKUP states --- each server applies the lease into the state it may allocate from --- and the leases are not placed on the queue for allocation to other clients. Instead they may only be found by client identity. The result is that the lease is only offered to the returning client.

Care should probably be taken to ensure that the client only has one lease within a given subnet that it is identified by.

Leases may be set *reserved* either through OMAPI, or through the **infinite-is-reserved** configuration option (if this is applicable to your environment and mixture of clients).

It should also be noted that leases marked *reserved* are effectively treated the same as leases marked *bootp*.

### 3.1.16 References

DHCP `option` statements are documented in the *dhcp-options (5)* manual page.

Expressions used in DHCP `option` statements and elsewhere are documented in the *dhcp-eval (5)* manual page.

### 3.1.17 Files

`/etc/lease/dhcpd.conf`

The configuration file for the *dhcpd(8)* program.

### 3.1.18 See also

*dhcpd(8)*, *dhcp-options(5)*, *dhcp-eval(5)*, *dhcpd.leases(5)*

### 3.1.19 Copyright

Copyright (C) 2025 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2004-2017 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1996-2003 by Internet Software Consortium.

## 3.2 dhcpd.leases --- DHCP lease database

### 3.2.1 Description

*dhcpd(8)* keeps a persistent database of leases that it has assigned. This database is a free-form ASCII file containing a series of lease declarations. Every time a lease is acquired, renewed or released, its new value is recorded at the end of the lease file. So if more than one declaration appears for a given lease, the last one in the file is the current one.

*dhcpd(8)* requires that a lease database file be present before it will start. To create the initial lease database, an empty file may be created at the path `/var/lib/lease/dhcpd.leases` using the *touch(1)* program:

```
# touch /var/lib/lease/dhcpd.leases
```

In order to prevent the lease database from growing without bound, the file is rewritten from time to time. First, a temporary lease database is created and all known leases are dumped to it. Then, the old lease database is renamed to `/var/lib/lease/dhcpd.leases~`. Finally, the newly written lease database is moved into place.

In order to process both DHCPv4 and DHCPv6 messages, two separate instances of the *dhcpd(8)* process will need to be run. Each of these instances will need its own lease file. **dhcpd's** `-lf` command line argument may be used to specify a different lease filename for one or both servers.

### 3.2.2 Format

Lease descriptions are stored in a format that is parsed by the same recursive descent parser used to read the *dhcpcd.conf(5)* and *dhclient.conf(5)* files. Lease files can contain lease declarations, and also group and subgroup declarations, host declarations and failover state declarations. Group, subgroup, and host declarations are used to record objects created using the OMAPI protocol.

The lease file is a log-structured file --- whenever a lease changes, the contents of that lease are written to the end of the file. This means that it is entirely possible and quite reasonable for there to be two or more declarations of the same lease in the lease file at the same time. In that case, the instance of that particular lease that appears last in the file is the one that is in effect.

Group, subgroup, and host declarations in the lease file are handled in a similar manner, except that if any of these objects are deleted, a rubout is written to the lease file. This is just the same declaration, with `{ deleted; }` in the scope of the declaration. When the lease file is rewritten, any such rubouts that can be eliminated are eliminated. It is possible to delete a declaration in the *dhcpcd.conf(5)* file; in this case, the rubout can never be eliminated from the `dhcpcd.leases` file.

### 3.2.3 Common statements for lease declarations

While the lease file formats for DHCPv4 and DHCPv6 are different they share many common statements and structures. This section describes the common statements while the succeeding sections describe the protocol specific statements.

#### 3.2.3.1 Dates

A date is specified in two ways, depending on the configuration value for the `db-time-format` parameter. If it is configured as `default`, then the date fields appear as follows:

```
<weekday> <year>/<month>/<day> <hour>:<minute>:<second>
```

The weekday is present to make it easy for a human to tell when a lease expires --- it is specified as a number from 0 to 6, with 0 being Sunday. The day of week is ignored on input. The year is specified with the century, so it should generally be four digits except for really long leases. The month is specified as a number starting with 1 for January. The day of the month is likewise specified starting with 1. The hour is a number between 0 and 23, the minute a number between 0 and 59, and the second also a number between 0 and 59.

Lease times are specified in Universal Coordinated Time (UTC), not in the local time zone. On most POSIX machines, the current time in UTC can be displayed by running the **date(1)** program:

```
# date -u
```

If the `db-time-format` is configured as `local`, then the date fields appear as follows:

```
epoch <seconds-since-epoch>; # <day-name> <month-name> <day-number>  
<hours>:<minutes>:<seconds> <year>
```

The `<seconds-since-epoch>` value is given according to the system's local clock (often referred to as "unix time"). The `#` symbol starts a comment that describes what actual time this is as according to the system's configured timezone, at the time the value was written. It is provided only for human inspection.

If a lease will never expire, date is written as the literal `never` instead of an actual date.

### 3.2.3.2 General variables

As part of the processing of a lease, information may be attached to the lease structure, for example, the DDNS information, or if a variable is specified in the configuration file. Some of these, like the DDNS information, have specific descriptions below. For others, such as any variables that are specified, a generic line of the following will be included.

```
set <variable> = <value>;
```

The **set** statement sets the value of a variable on the lease. For general information on variables, see the *dhcp-eval(5)* manual page.

### 3.2.3.3 DDNS Variables

#### **ddns-text**

This variable is used to record the value of the client's identification record when the server has updated DNS for a particular lease. The TXT record is used with the interim DDNS update style.

#### **ddns-dhcid**

This variable is used to record the value of the client's identification record when the server has updated DNS for a particular lease. The DHCID record is used for the standard DDNS update style.

#### **ddns-fwd-name**

This variable records the value of the DNS name used in updating the client's address record if a DDNS update has been successfully done by the server. The server may also have used this name to update the client's PTR record.

#### **ddns-client-fqdn**

If the server is configured both to use the interim or standard DDNS update style, and to allow clients to update their own FQDNs, then if the client did in fact update its own FQDN, the `ddns-client-fqdn` variable records

the DNS name that the client has indicated it is using. This is the name that the server will have used to update the client's PTR record in this case.

#### **ddns-rev-name**

If the server successfully updates the client's PTR record, this variable will record the DNS name that the DHCP server used for the PTR record. The DNS name to which the PTR record points will be either the `ddns-fwd-name` or the `ddns-client-fqdn`.

### **3.2.4 Executable statements**

**on** *<event>* [ | *<event>* ... ] { *<statement>* ... }

The **on** statement records a list of statements to execute if a certain *<event>* occurs. The possible events that can occur for an active lease are `release` and `expiry`. More than one event can be specified --- if so, the events are separated by | characters.

**authoring-byte-order** *<big-endian | little-endian>*;

This statement is automatically added to the top of new lease files by the server. It indicates the internal byte order of the server. This permits lease files generated on a server with one form of byte order to be read by a server with a different form. Lease files which do not contain this entry are simply treated as having the same byte order as the server reading them. If you are migrating lease files generated by a server that predates this statement and is of a different byte order than the your destination server, you can manually add this statement. It must proceed any lease entries. Valid values for this parameter are `little-endian` and `big-endian`.

### **3.2.5 The DHCPv4 lease declaration**

**lease** *<ip-address>* { *<statement>* ... }

Each lease declaration includes the single IP address that has been leased to the client. The statements within the braces define the duration of the lease and to whom it is assigned.

**starts** *<date>*;

Records the start time of a lease.

See the description of dates in the section titled *Common statements for lease declarations*.

**ends** *<date>*;

Records the end time of a lease.

See the description of dates in the section titled *Common statements for lease declarations*.

**tstp** <date>;

Present if the failover protocol is being used. Indicates what time the peer has been told the lease expires.

See the description of dates in the section titled *Common statements for lease declarations*.

**tsfp** <date>;

Present if the failover protocol is being used. Indicates the lease expiry time that the peer has acknowledged.

See the description of dates in the section titled *Common statements for lease declarations*.

**atsfp** <date>;

The actual time sent from the failover partner.

See the description of dates in the section titled *Common statements for lease declarations*.

**cltt** <date>;

The client's last transaction time.

See the description of dates in the section titled *Common statements for lease declarations*.

**hardware** <hardware-type> <mac-address>;

Records the MAC address of the network interface on which the lease will be used. It is specified as a series of hexadecimal octets, separated by colons.

**uid** <client-identifier>;

Records the client identifier used by the client to acquire the lease. Clients are not required to send client identifiers, and this statement only appears if the client did in fact send one. Client identifiers are normally an ARP type (1 for Ethernet) followed by the MAC address, just like in the **hardware** statement, but this is not required.

The client identifier is recorded as a colon-separated hexadecimal list or as a quoted string. If it is recorded as a quoted string and it contains one or more non-printable characters, those characters are represented as octal escapes --- a backslash character followed by three digits. The format used is determined by the lease-id-format parameter, which defaults to octal.

**client-hostname** <hostname>;

Most DHCP clients will send their hostname in the `host-name` option. If a client sends its hostname in this way, the hostname is recorded on the lease with a **client-hostname** statement. This is not required by the protocol, however, so many specialized DHCP clients do not send a `host-name` option.

**binding state** <state>;

The **binding state** statement declares the lease's binding state. When the DHCP server is not configured to use the failover protocol, a lease's binding state may be active, free or abandoned. The failover protocol adds some additional transitional states, as well as the backup state, indicates that the lease is available for allocation by the failover secondary. Please see *dhcpcd.conf*(5) for more information about abandoned leases.

**next binding state** <state>;

The **next binding state** statement indicates what state the lease will move to when the current state expires. The time when the current state expires is specified in the ends statement.

**rewind binding state** <state>;

This statement is part of an optimization for use with failover. This helps a server rewind a lease to the state most recently transmitted to its peer.

**option agent.circuit-id** <string>;

**option agent.remote-id** <string>;

These statements are used to record the circuit ID and remote ID options sent by the relay agent, if the relay agent uses the relay agent information option. This allows these options to be used consistently in conditional evaluations even when the client is contacting the server directly rather than through its relay agent.

**vendor-class-identifier** variable

The server retains the client-supplied Vendor Class Identifier option for informational purposes, and to render them in DHCPLEASEQUERY responses.

**bootp;**

Indicates that the BOOTP failover flag should be set. BOOTP dynamic leases are treated differently than normal dynamic leases, as they may only be used by the client to which they are currently allocated.

**reserved;**

Indicates that the RESERVED failover flag should be set. RESERVED dynamic leases are treated differently than normal dynamic leases, as they may only be used by the client to which they are currently allocated.

Additional options or executable statements may be included, see the description of them in the section titled *Common statements for lease declarations*.



### 3.2.6 The DHCPv6 lease (IA) declaration

**ia\_ta** <IAID\_DUID> { <statement> ... }

**ia\_na** <IAID\_DUID> { <statement> ... }

**ia\_pa** <IAID\_DUID> { <statement> ... }

Each lease declaration starts with a tag indicating the type of the lease. **ia\_ta** is for temporary addresses, **ia\_na** is for non-temporary addresses and **ia\_pd** is for prefix delegation. Following this tag is the combined IAID and DUID from the client for this lease.

The <IAID\_DUID> value is recorded as a colon-separated hexadecimal list or as a quoted string. If it is recorded as a quoted string and it contains one or more non-printable characters, those characters are represented octal escapes --- a backslash character followed by three octal digits. The format used is governed by the lease-id-format parameter, which defaults to octal.

**cltt** <date>;

The client's last transaction time.

See the description of dates in the section titled *Common statements for lease declarations*.

**iaaddr** <ipv6-address> { <statement> ... }

**iaprefix** <ipv6-address>/<prefix-length> { <statement> ... }

Within a given lease there can be multiple **iaaddr** and **iaprefix** statements. Each will have either an IPv6 address or an IPv6 prefix (an address and a prefix length indicating a CIDR style block of addresses). The following statements may occur within each **iaaddr** or **iaprefix**.

**binding state** <state>;

The **binding state** statement declares the lease's binding state. In DHCPv6, it will normally be active or expired.

**preferred-life** <lifetime>;

The IPv6 preferred lifetime associated with this address, in seconds.

**max-life** <lifetime>;

The valid lifetime associated with this address, in seconds.

**ends** <date>;

Records the end time of a lease.

See the description of dates in the section titled *Common statements for lease declarations*.

Additional options or executable statements may be included, see the description of them in the section titled *Common statements for lease declarations*.

### 3.2.7 The failover peer state declaration

The state of any failover peering arrangements is also recorded in the lease file, using the **failover peer** statement:

```
failover peer <name> state { my state <state> at <date>; peer state <state> at <date>; }
```

The states of the peer named name is being recorded. Both the state of the running server (**my state**) and the other failover partner (**peer state**) are recorded. The following states are possible: unknown-state, partner-down, normal, communications-interrupted, resolution-interrupted, potential-conflict, recover, recover-done, shutdown, paused, and startup.

### 3.2.8 Files

```
/var/lib/lease/dhcpd.leases
```

The DHCP leases file.

```
/var/lib/lease/dhcpd.leases~
```

Old DHCP leases file.

```
/var/lib/lease/dhcpd6.leases
```

The DHCPv6 leases file.

```
/var/lib/lease/dhcpd6.leases~
```

Old DHCPv6 leases file.

### 3.2.9 See also

*dhcpd(8)*, *dhcp-options(5)*, *dhcp-eval(5)*, *dhcpd.conf(5)*

### 3.2.10 Copyright

Copyright (C) 2025 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2004-2016 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1996-2003 by Internet Software Consortium.

## 3.3 `dhclient.conf` --- DHCP client configuration

### 3.3.1 Description

`dhclient.conf` is the configuration file for `dhclient(8)`, the DHCP client program.

`dhclient.conf` is a free-form ASCII text file. It is parsed by the recursive-descent parser built into `dhclient(8)`. The file may contain extra tabs and newlines for formatting purposes. Keywords in the file are case-insensitive. Comments may be placed anywhere within the file, except within quotes. Comments begin with the `#` character and end at the end of the line.

The `dhclient.conf` file can be used to configure the behaviour of the client in a wide variety of ways: protocol timing, information requested from the server, information required of the server, defaults to use if the server does not provide certain information, values with which to override information provided by the server, or values to prepend or append to information provided by the server. The configuration file can also be preinitialized with addresses to use on networks that don't have DHCP servers.

### 3.3.2 Protocol timing

The timing behaviour of the client need not be configured by the user. If no timing configuration is provided by the user, a fairly reasonable timing behaviour will be used by default --- one which results in fairly timely updates without placing an inordinate load on the server.

If required the following statements can be used to adjust the timing behaviour of the DHCPv4 client. The DHCPv6 protocol provides values to use and they are not currently configurable for DHCPv6.

**timeout** *<time>*;

The **timeout** statement determines the amount of time that must pass between the time that the client begins to try to determine its address the time that it decides that it's not going to be able to contact a server. By default, this timeout is 60 seconds. After the timeout has passed, if there are any static leases defined in the configuration, or any leases remaining in the lease database that have not yet expired, the client will loop through these leases attempting to validate them, and if it finds one that appears to be valid, it will use lease's address. If there are no valid static leases or unexpired leases in the lease database, the client will restart the protocol after the defined retry interval.

**retry** *<time>*;

The **retry** statement determines the time that must pass after the client has determined that there is no DHCP server present before it tries again to contact a DHCP server. By default, this is 5 minutes.

**select-timeout** *<time>;*

It is possible (some might say desirable) for there to be more than one DHCP server serving any given network. In this case, it is possible that a client may be sent more than one offer in response to its initial lease discovery message. It may be that one of these offers is preferable to the other (e.g., one offer may have the address the client previously used, and the other may not).

**select-timeout** configures the time after the client sends its first lease discovery request at which it stops waiting for offers from servers, assuming that it has received at least one such offer. If no offers have been received by the time the select-timeout has expired, the client will accept the first offer that arrives. By default, the timeout is 0 seconds, i.e., the client will take the first offer it sees.

**reboot** *<time>;*

When the client is restarted, it first tries to reacquire the last address it had. This is called the INIT-REBOOT state. If it is still attached to the same network it was attached to when it last ran, this is the quickest way to get started. The **reboot** statement sets the time that must elapse after the client first tries to reacquire its old address before it gives up and tries to discover a new address. By default, the reboot timeout is 10 seconds.

**backoff-cutoff** *<time>;*

The client uses an exponential backoff algorithm with some randomness, so that if many clients try to configure themselves at the same time, they will not make their requests in lockstep. The **backoff-cutoff** statement determines the maximum amount of time that the client is allowed to back off, the actual value will be evaluated randomly between 1/2 to 1 1/2 times the *<time>* specified. It defaults to 15 seconds.

**initial-interval** *<time>;*

The **initial-interval** statement sets the amount of time between the first attempt to reach a server and the second attempt to reach a server. Each time a message is sent, the interval between messages is incremented by the current interval multiplied by a random number between 0 and 1. If it is greater than the **backoff-cutoff** amount, it is set to that amount. It defaults to 10 seconds.

**initial-delay** *<time>;*

The **initial-delay** statement sets the maximum time client can wait after start before commencing first transmission. According to [RFC 2131](#) section 4.4.1, a client should wait for a random time between startup and the actual first transmission. It defaults to 0 seconds.

### 3.3.3 DHCPv6 lease selection

In the DHCPv6 protocol the client will wait a small amount of time to allow ADVERTISE messages from multiple servers to arrive. It will then need to choose from all of the messages that may have arrived before proceeding to making a request of the selected server.

The first selection criteria is the set of options and addresses in the message. Messages that don't include an option specified as required will be given a score of 0 and not used. If the **dhclient(8)** `-R` command line argument is used, then messages that don't include the correct number of bindings (IA-NA, IA-TA or IA-PD) will be discarded.

The next criteria is the preference value from the message, with the highest preference value being used even if leases with better addresses or options are available.

Finally the lease is scored and the lease with the highest score is selected. A lease's score is based on the number of bindings, number of addresses and number of options it contains:

```
bindings * X + addresses * Y + options
```

By default  $X=10000$  and  $Y=100$ . This will cause the client to select a lease with more bindings, over a lease with less bindings but more addresses. The weightings were changed as part of implementing 7550. Previously they were  $X=50$  and  $Y=100$  meaning more addresses were preferred over more bindings.

### 3.3.4 Lease requirements and requests

The DHCP protocol allows the client to request that the server send it specific information, and not send it other information that it is not prepared to accept. The protocol also allows the client to reject offers from servers if they don't contain information the client needs, or if the information provided is not satisfactory.

There is a variety of data contained in offers that DHCP servers send to DHCP clients. The data that can be specifically requested are called DHCP options. DHCP options are defined in *dhcp-options(5)*.

The following are lease requirement and request related statements:

```
[ also ] request [ [ <option-space> . ] <option> ] [, ... ];
```

The **request** statement causes the client to request that any server responding to the client send the client its values for the specified options. Only the option names should be specified in the request statement --- not option parameters. By default, the DHCPv4 client requests the `subnet-mask`, `broadcast-address`, `time-offset`, `routers`, `domain-name`, `domain-name-servers`, and `host-name` options while the DHCPv6 client requests the `dhcp6.name-servers` and `dhcp6.domain-search` options.

**Note:** If a **request** statement is specified, these defaults are overridden and these options will not be requested.

In some cases, it may be desirable to send no parameter request list at all. This can be configured by using the following statement without any arguments:

```
request;
```

In most cases, it is desirable to simply add one option to the request list which is of interest to the client in question. In this case, the **also** keyword may be used. For example:

```
also request domain-search, dhcp6.sip-servers-addresses;
```

[ **also** ] **require** [ [ *<option-space>* . ] *<option>* ] [ , ... ];

The **require** statement lists options that must be sent in order for an offer to be accepted. Offers that do not contain all the listed options will be ignored. There is no default require list.

An example of using the **require** statement follows:

```
require name-servers;

interface eth0 {
    also require domain-search;
}
```

**send** *<option declaration>*;

The **send** statement causes the client to send the specified option to the DHCP server with the specified value. This is a full option declaration as described in *dhcp-options(5)*. Options that are always sent in the DHCP protocol should not be specified here, except that the client can specify a requested *dhcp-lease-time* option other than the default requested lease time, which is 2 hours. The other obvious use for this statement is to send information to the server that will allow it to differentiate between this client and other clients or kinds of clients.

### 3.3.5 Dynamic DNS updates

*dhclient(8)* now has some very limited support for doing DNS updates when a lease is acquired. Note that everything in this section is true whether DHCPv4 or DHCPv6 is used. The exact same syntax is used for both.

A key and zone have to be declared, as in the DHCP server (see *dhcpd.conf(5)* for details). The *fqdn* option also has to be configured on the client as follows:

```
send fqdn.fqdn "grosse.example.com.";
send fqdn.encoded on;
send fqdn.server-update off;
also request fqdn, dhcp6.fqdn;
```

The `fqdn.fqdn` option MUST be a fully-qualified domain name. A zone statement must be defined for the zone that is to be updated. The `fqdn.encoded` option may need to be set to `on` or `off`, depending on the DHCP server that is used.

The following are DDNS related statements:

**do-forward-updates** <flag>;

If DNS updates must be done in the DHCP client script (see *dhclient-script*(8)) rather than having the DHCP client do the update directly --- for example, if SIG(0) authentication has to be used, which is not supported directly by the DHCP client --- the DHCP client can be configured not to do the DNS update using the **do-forward-updates** statement. <flag> should be true if the DHCP client must do the DNS update, and false if the DHCP client must NOT do the DNS update. The default is true, i.e., *dhclient*(8) will do the DNS update.

### 3.3.6 Option modifiers

In some cases, a DHCP client may receive option data from the DHCP server which is not really appropriate for that client, or may not receive information that it needs, and for which a useful default value exists. It may also receive information which is useful, but which needs to be supplemented with local information. To handle these needs, several option modifiers statements are available:

**default** <option declaration>;

If for some option the client should use the value supplied by the server, but needs to use some default value if no value was supplied by server, these values can be defined in the default statement.

**supersede** <option declaration>;

If for some option the client should always use a locally-configured or values rather than whatever is supplied by the server, these can be defined in the **supersede** statement.

**prepend** <option declaration>;

If for some set of options the client should use a value you supply, and then use the values supplied by the server, if any, these values can be defined in the **prepend** statement. The **prepend** statement can only be used for options which allow more than one value to be given. This restriction is not enforced --- if you ignore it, the behaviour will be unpredictable.

**append** <option declaration>;



If for some set of options the client should first use the values supplied by the server, if any, and then use values you supply, these values can be defined in the **append** statement. The **append** statement can only be used for options which allow more than one value to be given. This restriction is not enforced --- if you ignore it, the behaviour will be unpredictable.

### 3.3.7 Lease declarations

```
lease { <lease-declaration> [ ... <lease-declaration> ] }
```

The DHCP client may decide after some period of time (see the section on *Protocol timing*) that it is not going to succeed in contacting a server. At that time, it consults its own database of old leases and tests each one that has not yet timed out by pinging the listed router for that lease to see if that lease could work. It is possible to define one or more fixed leases in the client configuration file for networks where there is no DHCP or BOOTP service, so that the client can still automatically configure its address. This is done with the **lease** statement.

---

**Note:** The **lease** statement is also used in the *dhclient.leases(5)* file in order to record leases that have been received from DHCP servers. Some of the syntax for leases as described below is only needed in the *dhclient.leases(5)* file. Such syntax is documented here for completeness.

---

A lease statement consists of the **lease** keyword, followed by a left curly brace (**{**), followed by one or more lease declaration statements, followed by a right curly brace (**}**). The following lease declarations are possible:

#### **bootp;**

The **bootp** statement is used to indicate that the lease was acquired using the BOOTP protocol rather than the DHCP protocol. It is never to specify this in the client configuration file. The client uses this syntax in its lease database file.

#### **interface <string>;**

The **interface** lease statement is used to indicate the interface on which the lease is valid. If set, this lease will only be tried on a particular interface. When the client receives a lease from a server, it always records the interface number on which it received that lease. If predefined leases are specified in the *dhclient.conf* file, the interface should also be specified, although this is not required.

#### **fixed-address <ip-address>;**

The **fixed-address** statement is used to set the IP address of a particular lease. This is required for all **lease** statements. The IP address must be specified as a dotted quad (e.g., 12.34.56.78).

#### **filename <string>;**



The **filename** statement specifies the name of the boot filename to use. This is not used by the standard client configuration script, but is included for completeness.

**server-name** <string>;

The **server-name** statement specifies the name of the boot server name to use. This is also not used by the standard client configuration script.

**option** <option-declaration>;

The **option** statement is used to specify the value of an option supplied by the server, or, in the case of predefined leases declared in `dhclient.conf`, the value that the user wishes the client configuration script to use if the predefined lease is used.

**script** <script-name>;

The **script** statement is used to specify the pathname of the DHCP client configuration script (`dhclient-script(8)`). This script is used by the DHCP client to set each interface's initial configuration prior to requesting an address, to test the address once it has been offered, and to set the interface's final configuration once a lease has been acquired. If no lease is acquired, the script is used to test predefined leases, if any, and also called once if no valid lease can be identified. For more information, see `dhclient-script(8)`.

**vendor option space** <string>;

The *vendor option space*\* statement is used to specify the name of the option space which should be used for decoding the **vendor-encapsulate-options** option if one is received. The **dhcp-vendor-identifier** can be used to request a class of vendor options from the DHCP server. See `dhcp-options(5)` for details.

**medium** <media setup string>;

The **medium** statement can be used on systems where network interfaces cannot automatically determine the type of network to which they are connected. The <media setup string> argument is a system-dependent parameter which is passed to the DHCP client configuration script (`dhclient-script(8)`) when initializing the interface. The argument is passed on the `ifconfig(8)` program's command line when configuring the interface.

The DHCP client automatically declares this parameter if it uses a media type (see the [media](#) statement) when configuring the interface in order obtain a lease. This statement should be used in predefined leases if the network interface requires media type configuration.

**renew** <date>;

**rebind** <date>;

**expire** <date>;

The **renew** statement defines the time at which the DHCP client should begin trying to contact its server to renew a lease that it is using.

The **rebind** statement defines the time at which the DHCP client should begin to try to contact any DHCP server in order to renew its lease.

The **expire** statement defines the time at which the DHCP client must stop using a lease if it has not been able to contact a server in order to renew it.

These declarations are automatically set in leases acquired by the DHCP client, but must also be configured in predefined leases --- a predefined lease whose expiry time has passed will not be used by the DHCP client.

Dates are specified in one of two ways. The software will output times in these two formats depending on if the `db-time-format` configuration parameter has been set to `default` or `local`. If it is set to `default`, then date values appear as follows:

<weekday> <year>/<month>/<day> <hour>:<minute>:<second>

The weekday is present to make it easy for a human to tell when a lease expires --- it is specified as a number from 0 to 6, with 0 being Sunday. The day of week is ignored on input. The year is specified with the century, so it should generally be four digits except for really long leases. The month is specified as a number starting with 1 for January. The day of the month is likewise specified starting with 1. The hour is a number between 0 and 23, the minute a number between 0 and 59, and the second also a number between 0 and 59.

If the `db-time-format` was configured to `local`, then the date fields appear as follows:

**epoch** <seconds-since-epoch>; # <day-name> <month-name> <day-number> <hours>:<minutes>:<seconds> <year>

The <seconds-since-epoch> value is as according to the system's local clock (often referred to as "unix time"). The # symbol starts a comment that describes what actual time this is as according to the system's configured timezone, at the time the value was written. It is provided only for human inspection. The epoch time is the only recommended value for machine inspection.

If a lease will never expire, then the date is the literal `never` instead of an actual date.

---

**Note:** When defining a static lease, one may use either time format one wishes, and need not include the comment or values after it.

---

### 3.3.8 Alias declarations

**alias** { <declaration> ... }

Some DHCP clients running TCP/IP roaming protocols may require that in addition to the lease they may acquire via DHCP, their interface also be configured with a pre-defined IP address alias so that they can have a permanent IP address even while roaming. *dhclient(8)* doesn't support roaming with fixed addresses directly, but in order to facilitate such experimentation, it can be set up to configure an IP address alias using the **alias** declaration.

The **alias** declaration resembles a **lease** declaration, except that options other than the subnet-mask option are ignored by the standard client configuration script (*dhclient-script(8)*), and expiry times are ignored. A typical **alias** declaration includes an interface declaration, a fixed-address declaration for the IP address alias, and a subnet-mask option declaration. A **medium** statement should never be included in an **alias** declaration.

### 3.3.9 Other declarations

**db-time-format** (*default* | *local*);

The **db-time-format** option determines which of two output methods are used for printing times in leases files. The *default* format provides day-and-time in UTC, whereas *local* uses a seconds-since-epoch to store the time value, and helpfully places a local timezone time in a comment on the same line. The formats are described in detail in the section on [Lease declarations](#).

**lease-id-format** (*octal* | *hex*);

This parameter governs the format used to write certain values to lease files. With the default format *octal*, values are written as quoted strings in which non-printable characters are represented as octal escapes --- a backslash character followed by three octal digits. When the *hex* format is specified, values are written as an unquoted series of hexadecimal digit pairs, separated by colons.

Currently, the values written out based on **lease-id-format** are the *default-duid* and the IAID value (DHCPv6 only). The client automatically reads the values in either format.

---

**Note:** When the format is *octal*, rather than as an octal string, IAID is output as hex if it contains no printable characters or as a string if contains only printable characters. This is done to maintain backward compatibility.

---

**reject** <cidr-ip-address> [, ... <cidr-ip-address> ];

The **reject** statement causes the DHCP client to reject offers from DHCP servers whose server identifier matches any of the specified hosts or sub-

nets. This can be used to avoid being configured by rogue or misconfigured DHCP servers, although it should be a last resort --- better to track down the bad DHCP server and fix it.

The **cidr-ip-address** configuration type is of the form `<ip-address>[/*<prefixlen>]`, where `<ip-address>` is a dotted quad IP address, and the optional `<prefixlen>` is the CIDR prefix length of the subnet, counting the number of significant bits in the netmask starting from the leftmost end.

The following is an example:

```
reject 192.168.0.0/16, 10.0.0.5;
```

The above example would cause offers from any server identifier in the entire [RFC 1918](#) "Class C" network 192.168.0.0/16, or the specific single address 10.0.0.5, to be rejected.

**interface** `<name>` { `<declaration>` ... }

A client with more than one network interface may require different behaviour depending on which interface is being configured. All timing parameters and declarations other than **lease** and **alias** can be enclosed in an **interface** declaration, and those parameters will then be used only for the interface that matches the specified `<name>`. Interfaces for which there is no interface declaration will use the parameters declared outside of any interface declaration, or the default settings.

---

**Note:** *dhclient(8)* only maintains one list of interfaces, which is either determined at startup from command line arguments, or otherwise is autodetected. If you supplied the list of interfaces on the command line, this configuration clause will add the named interface to the list in such a way that will cause it to be configured by DHCP. It may not be the result you had intended. This is an undesirable side effect that will be addressed in a future release.

---

**pseudo** `<name>` `<real-name>` { `<declaration>` ... }

Under some circumstances it can be useful to declare a pseudo-interface and have the DHCP client acquire a configuration for that interface. Each interface that the DHCP client is supporting normally has a DHCP client state machine running on it to acquire and maintain its lease. A pseudo-interface is just another state machine running on the interface named `<real-name>`, with its own lease and its own state. If you use this feature, you must provide a client identifier for both the pseudo-interface and the actual interface, and the two identifiers must be different. You must also provide a separate client script (*dhclient-script(8)*) for the pseudo-interface to do what you want with the IP address.

For example:

```
interface "ep0" {
    send dhcp-client-identifier "my-client-ep0";
}
pseudo "secondary" "ep0" {
    send dhcp-client-identifier "my-client-ep0-secondary";
    script "/etc/lease/dhclient-secondary";
}
```

The client script for the pseudo-interface should not configure the interface up or down --- essentially, all it needs to handle are the states where a lease has been acquired or renewed, and the states where a lease has expired. See *dhclient-script(8)* for more information.

**media** <media setup string> [, <media setup string>, ... ];

The **media** statement defines one or more media configuration parameters which may be tried while attempting to acquire an IP address. The DHCP client will cycle through each <media setup string> on the list, configuring the interface using that setup and attempting to boot, and then trying the next one. This can be used for network interfaces which aren't capable of sensing the media type unaided --- whichever media type succeeds in getting a request to the server and hearing the reply is probably right (no guarantees).

The media setup is only used for the initial phase of address acquisition (the DHCPDISCOVER and DHCPOFFER packets). Once an address has been acquired, the DHCP client will record it in its lease database and will record the media type used to acquire the address. Whenever the client tries to renew the lease, it will use that same media type. The lease must expire before the client will go back to cycling through media types.

**hardware** <link-type> <mac-address>;

The **hardware** statement defines the hardware MAC address to use for this interface, for DHCP servers or relays to direct their replies. *dhclient(8)* will determine the interface's MAC address automatically, so use of this parameter is not recommended. The <link-type> corresponds to the interface's link layer type (e.g., ethernet), and the <mac-address> is a string of colon-separated hexadecimal values for octets.

**anycast-mac** <link-type> <mac-address>;

The **anycast-mac** statement overrides the all-ones broadcast MAC address *dhclient(8)* will use when it is transmitting packets to the all-ones limited broadcast IPv4 address. This configuration parameter is useful to reduce the number of broadcast packets transmitted by DHCP clients, but is only useful if you know the DHCP service(s) anycast MAC address prior to configuring your client. The <link-type> and <mac-address> parameters are configured in a similar manner to the **hardware** statement.

### 3.3.10 Example

The following configuration file is an example for use on an imaginary laptop. The laptop has an IP address alias of 192.5.5.213, and has one interface `ep0` (a 3com 3C589C). Booting intervals have been shortened somewhat from the default, because the client is expected to spend most of its time on networks with little DHCP activity. The laptop is expected to roam to multiple networks.

```
timeout 60;
retry 60;
reboot 10;
select-timeout 5;
initial-interval 2;
reject 192.33.137.209;

interface "ep0" {
    send host-name "laptop.example.com";
    hardware ethernet 00:a0:24:ab:fb:9c;
    send dhcp-client-identifier 1:0:a0:24:ab:fb:9c;
    send dhcp-lease-time 3600;
    supersede domain-search
        "example.com", "work.example.org", "home.example.org";
    prepend domain-name-servers 127.0.0.1;
    request subnet-mask, broadcast-address, time-offset, routers,
        domain-name, domain-name-servers, host-name;
    require subnet-mask, domain-name-servers;
    script "/usr/sbin/dhclient-script";
    media "media 10baseT/UTP", "media 10base2/BNC";
}

alias {
    interface "ep0";
    fixed-address 192.5.5.213;
    option subnet-mask 255.255.255.255;
}
```

This is a very complicated `dhclient.conf` file --- in general, yours should be much simpler. In many cases, it's sufficient to just create an empty `dhclient.conf` file --- the defaults are usually fine.

### 3.3.11 Files

`/etc/lease/dhclient.conf`

The configuration file for the *dhclient* (8) program.

### 3.3.12 See also

*dhclient(8)*, *dhcp-options(5)*, *dhcp-eval(5)*, *dhclient.leases(5)*,  
*dhcpd(8)*, *dhcpd.conf(5)*

### 3.3.13 Copyright

Copyright (C) 2025 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2004-2016 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1996-2003 by Internet Software Consortium.

## 3.4 *dhclient.leases* --- DHCP client lease database

### 3.4.1 Description

*dhclient(8)* keeps a persistent database of leases that it has acquired that are still valid. This database is a free-form ASCII file containing one valid declaration per lease. The file is written as a log, so it is not unusual to find multiple declarations for the same lease. If more than one declaration appears for a given lease, the last one in the file is used.

The format of the lease declarations is described in *dhclient.conf(5)*.

### 3.4.2 Files

`/var/lib/lease/dhclient.leases`

The client leases file.

### 3.4.3 See also

*dhclient(8)*, *dhcp-options(5)*, *dhcp-eval(5)*, *dhclient.conf(5)*,  
*dhcpd(8)*, *dhcpd.conf(5)*

### 3.4.4 Copyright

Copyright (C) 2025 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2009-2011 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 2004 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1997-2003 by Internet Software Consortium.

## 3.5 `dhcp-eval` --- DHCP conditional evaluation

### 3.5.1 Description

`dhcpd(8)` and `dhclient(8)` both provide the ability to perform conditional behavior depending on the contents of packets they receive. The syntax for specifying this conditional behaviour is documented here.

### 3.5.2 Conditional behavior

Conditional behaviour may be specified using the `if` and `switch` statements. A conditional statement can appear anywhere that a regular statement (e.g., an `option` statement) can appear, and can enclose one or more such statements.

#### 3.5.2.1 The `if` statement

A typical conditional `if` statement in a server might be:

```
if option dhcp-user-class = "accounting" {
    max-lease-time 17600;
    option domain-name "accounting.example.org";
    option domain-name-servers ns1.accounting.example.org,
                               ns2.accounting.example.org;
} elsif option dhcp-user-class = "sales" {
    max-lease-time 17600;
    option domain-name "sales.example.org";
    option domain-name-servers ns1.sales.example.org,
                               ns2.sales.example.org;
} elsif option dhcp-user-class = "engineering" {
    max-lease-time 17600;
    option domain-name "engineering.example.org";
    option domain-name-servers ns1.engineering.example.org,
                               ns2.engineering.example.org;
} else {
    max-lease-time 600;
    option domain-name "misc.example.org";
```

(continues on next page)



(continued from previous page)

```
option domain-name-servers ns1.misc.example.org,
                           ns2.misc.example.org;
}
```

On the client side, an example of conditional evaluation might be:

```
# example.org filters DNS at its firewall, so we have to use their
↳DNS
# servers when we connect to their network.  If we are not at
# example.org, prefer our own DNS server.
if not option domain-name = "example.org" {
    prepend domain-name-servers 127.0.0.1;
}
```

The **if** statement and the **elsif** continuation statement both take boolean expressions as their arguments. That is, they take expressions that, when evaluated, produce a boolean result. If the expression evaluates to true, then the statements enclosed in braces following the **if** statement are executed, and all subsequent **elsif** and **else** clauses are skipped. Otherwise, each subsequent **elsif** clause's expression is checked, until an **elsif** clause is encountered whose test evaluates to true. If such a clause is found, the statements in braces following it are executed, and then any subsequent **elsif** and **else** clauses are skipped. If all the **if** and **elsif** clauses are checked but none of their expressions evaluate true, then if there is an **else** clause, the statements enclosed in braces following the **else** are evaluated. Boolean expressions that evaluate to null are treated as false in conditionals.

### 3.5.2.2 The **switch** statement

The previous example can be rewritten using a **switch** construct:

```
switch (option dhcp-user-class) {
    case "accounting":
        max-lease-time 17600;
        option domain-name "accounting.example.org";
        option domain-name-servers ns1.accounting.example.org,
                                   ns2.accounting.example.org;

    case "sales":
        max-lease-time 17600;
        option domain-name "sales.example.org";
        option domain-name-servers ns1.sales.example.org,
                                   ns2.sales.example.org;

        break;
    case "engineering":
        max-lease-time 17600;
        option domain-name "engineering.example.org";
        option domain-name-servers ns1.engineering.example.org,
                                   ns2.engineering.example.org;
}
```

(continues on next page)

(continued from previous page)

```
break;
default:
    max-lease-time 600;
    option domain-name "misc.example.org";
    option domain-name-servers ns1.misc.example.org,
                                ns2.misc.example.org;
    break;
}
```

The **switch** statement and the **case** statements can both be data expressions or numeric expressions. Within a **switch** statement they all must be the same type. The server evaluates the expression from the **switch** statement and then it evaluates the expressions from the **case** statements until it finds a match.

If it finds a match it starts executing statements from that **case** until the next **break** statement. If it doesn't find a match it starts executing statements from the **default** statement and continues till the next **break** statement. If there is no match and there is no **default** statement, it does nothing.

### 3.5.3 Boolean expressions

The following is the current list of boolean expressions that are supported by the DHCP distribution.

*<data-expression-1> = <data-expression-2>*

The **=** operator compares the values of two data expressions, returning true if they are the same, false if they are not. If either the left-hand side or the right-hand side are null, the result is also null.

*<data-expression-1> ~= <data-expression-2>*

*<data-expression-1> ~~ <data-expression-2>*

The **~=** and **~~** operators perform extended *regex* (7) matching of the values of two data expressions, returning true if *<data-expression-1>* matches against the regular expression evaluated by *<data-expression-2>*, or false if it does not match or encounters some error. If either the left-hand side or the right-hand side are null or empty strings, the result is also false. The **~=** operator is case-sensitive, whereas the **~~** operator is case-insensitive.

*<boolean-expression-1> and <boolean-expression-2>*

The **and** operator evaluates to true if the boolean expression on the left-hand side and the boolean expression on the right-hand side both evaluate to true. Otherwise, it evaluates to false. If either the expression on the left-hand side or the expression on the right-hand side are null, the result is null.

*<boolean-expression-1> or <boolean-expression-2>*

The **or** operator evaluates to true if either the boolean expression on the left-hand side or the boolean expression on the right-hand side evaluate to true. Otherwise, it evaluates to false. If either the on the left-hand side or the expression on the right-hand side are null, the result is null.

**not** *<boolean-expression>*

The **not** operator evaluates to true if *<boolean-expression>* evaluates to false, and returns false if *<boolean-expression>* evaluates to true. If *<boolean-expression>* evaluates to null, the result is also null.

**exists** *<option-name>*

The **exists** expression returns true if the specified option exists in the incoming DHCP packet being processed.

**known**

The **known** expression returns true if the client whose request is currently being processed is known, i.e., if there's a host declaration for it.

**static**

The **static** expression returns true if the lease assigned to the client whose request is currently being processed is derived from a static address assignment.

### 3.5.4 Data expressions

Several of the boolean expressions above depend on the results of evaluating data expressions. A list of these expressions is provided here.

**substring** (*<data-expression>*, *<offset>*, *<length>*)

The **substring** operator evaluates *<data-expression>* and returns the substring of the result of that evaluation that starts *<offset>* bytes from the beginning, continuing for *<length>* bytes. *<offset>* and *<length>* are both numeric expressions. If *<data-expression>*, *<offset>*, or *<length>* evaluate to null, then the result is also null. If offset is greater than or equal to the length of the evaluated data, then a zero-length data string is returned. If length is greater then the remaining length of the evaluated data after offset, then a data string containing all data from offset to the end of the evaluated data is returned.

**suffix** (*<data-expression>*, *<length>*)

The **suffix** operator evaluates *<data-expression>* and returns the last *<length>* bytes of the result of that evaluation. *<length>* is a numeric expression. If *<data-expression>* or *<length>* evaluate to null, then the result is also null. If **suffix** evaluates to a number greater than the length of the evaluated data, then the evaluated data is returned.

**lcase** (*<data-expression>*)

The **lcase** function returns the result of evaluating *<data-expression>* converted to lower case. If *<data-expression>* evaluates to null, then the result is also null.

**ucase** (*<data-expression>*)

The **ucase** function returns the result of evaluating *<data-expression>* converted to upper case. If *<data-expression>* evaluates to null, then the result is also null.

**option** *<option-name>*

The **option** operator returns the contents of the specified option in the packet to which the server is responding.

**config-option** *<option-name>*

The **config-option** operator returns the value for the specified option that the DHCP client or server has been configured to send.

**gethostname()**

The **gethostname()** function returns a data string whose contents are a character string, the results of calling **gethostname()** on the local system with a size limit of 255 bytes (not including NULL terminator). This can be used for example to configure *dhclient(8)* to send the local hostname without knowing the local hostname at the time *dhclient.conf(5)* is written.

**hardware**

The **hardware** operator returns a data string whose first element is the type of network interface indicated in packet being considered, and whose subsequent elements are client's link-layer address. If there is no packet, or if the **RFC 2131** *hlen* field is invalid, then the result is null. Hardware types include Ethernet (1). Hardware types are specified by the IETF, and details on how the type numbers are defined can be found in **RFC 2131**.

**packet** (*<offset>*, *<length>*)

The **packet** operator returns the specified portion of the packet being considered, or null in contexts where no packet is being considered. *<offset>* and *<length>* are applied to the contents packet as in the **substring** operator.

"*<string>*"

A string, enclosed in quotes, may be specified as a data expression, and returns the text between the quotes, encoded in ASCII. The backslash (\) character is treated specially, as in C programming: \t means TAB, \r means carriage return, \n means newline, and \b means bell. Any octal value can be specified with \nnn, where *<nnn>* is any positive octal number less than 0400. Any hexadecimal value can be specified with 'xnn', where *<nn>* is any positive hexadecimal number less than or equal to 0xff.

*<colon-separated-hexadecimal-list>*

A list of hexadecimal octet values, separated by colons, may be specified as a data expression.

**concat** (<data-expression-1>, ..., <data-expression-N>)

The expressions are evaluated, and the results of each evaluation are concatenated in the sequence that the subexpressions are listed. If any subexpression evaluates to null, the result of the concatenation is null.

**reverse** (<numeric-expression>, <data-expression>)

The two expressions are evaluated, and then the result of evaluating the <data-expression> is reversed in place, using hunks of the size specified in the <numeric-expression>. For example, if <numeric-expression> evaluates to 4, and <data-expression> evaluates to 12 bytes of data, then the reverse expression will evaluate to 12 bytes of data, consisting of the last 4 bytes of the input data, followed by the middle 4 bytes, followed by the first 4 bytes.

**leased-address**

In any context where the client whose request is being processed has been assigned an IP address, this data expression returns that IP address. In any context where the client whose request is being processed has not been assigned an IP address, if this data expression is found in executable statements executed on that client's behalf, a log message indicating "there is no lease associated with this client" is syslogged at the debug level (this is considered *dhcpcd.conf*(5) debugging information).

**binary-to-ascii** (<numeric-expression-1>, <numeric-expression-2>, <data-expression-1>, <data-expression-2>)

Converts the result of evaluating <data-expression-2> into a text string containing one number for each element of the result of evaluating <data-expression-2>. Each number is separated from the other by the result of evaluating <data-expression-1>. The result of evaluating <numeric-expression-1> specifies the base (2 through 16) into which the numbers should be converted. The result of evaluating <numeric-expression-2> specifies the width in bits of each number, which may be either 8, 16, or 32.

As an example of the preceding three types of expressions, to produce the name of a PTR record for the IP address being assigned to a client, one could write the following expression:

```
concat (binary-to-ascii (10, 8, ".",  
                        reverse (1, leased-address)),  
        ".in-addr.arpa.");
```

**encode-int** (<numeric-expression>, <width>)

<numeric-expression> is evaluated and encoded as a data string of the specified width, in network byte order (most significant byte first). If the expression evaluates to the null value, the result is also null.

**pick-first-value** (<data-expression-1> [ , ... <data-expression-N> ])

The **pick-first-value** function takes any number of data expressions as its arguments. Each expression is evaluated, starting with the first in the list, until an expression is found that does not evaluate to a null value. That expression is returned, and none of the subsequent expressions are evaluated. If all expressions evaluate to a null value, the null value is returned.

**host-decl-name**

The **host-decl-name** function returns the name of the host declaration that matched the client whose request is currently being processed, if any. If no host declaration matched, the result is the null value.

### 3.5.5 Numeric expressions

Numeric expressions are expressions that evaluate to an integer. In general, the maximum size of such an integer should not be assumed to be representable in fewer than 32 bits, but the precision of such integers may be more than 32 bits.

In addition to the following operators several standard math functions are available. They are:

Operation	Symbol
Add	+
Subtract	−
Multiply	*
Divide	/
Modulus	%
Bitwise AND	&
Bitwise OR	
Bitwise XOR	^

**extract-int** (<data-expression>, <width>)

The **extract-int** operator extracts an integer value in network byte order from the result of evaluating the specified <data-expression>. <width> is the width in bits of the integer to extract. Currently, the only supported widths are 8, 16 and 32. If the evaluation of <data-expression> doesn't provide sufficient bits to extract an integer of the specified size, the null value is returned.

**lease-time**

The duration of the current lease, i.e., the difference between the current time and the time that the lease expires.

<number>

Any number between zero and the maximum representable size may be specified as a numeric expression.

## client-state

The current state of the client instance being processed. This is only useful in DHCP client configuration files. Possible values are:

- *Booting* --- DHCP client is in the INIT state, and does not yet have an IP address. The next message transmitted will be a DHCPDISCOVER, which will be broadcast.
- *Reboot* --- DHCP client is in the INIT-REBOOT state. It has an IP address, but is not yet using it. The next message to be transmitted will be a DHCPREQUEST, which will be broadcast. If no response is heard, the client will bind to its address and move to the BOUND state.
- *Select* --- DHCP client is in the SELECTING state --- it has received at least one DHCPOFFER message, but is waiting to see if it may receive other DHCPOFFER messages from other servers. No messages are sent in the SELECTING state.
- *Request* --- DHCP client is in the REQUESTING state --- it has received at least one DHCPOFFER message, and has chosen which one it will request. The next message to be sent will be a DHCPREQUEST message, which will be broadcast.
- *Bound* --- DHCP client is in the BOUND state --- it has an IP address. No messages are transmitted in this state.
- *Renew* --- DHCP client is in the RENEWING state --- it has an IP address, and is trying to contact the server to renew it. The next message to be sent will be a DHCPREQUEST message, which will be unicast directly to the server.
- *Rebind* --- DHCP client is in the REBINDING state --- it has an IP address, and is trying to contact any server to renew it. The next message to be sent will be a DHCPREQUEST, which will be broadcast.

## 3.5.6 Action expression

**log** (<priority>, <data-expression>)

Logging statements may be used to send information to the standard logging channels. A logging statement includes an optional <priority> (fatal, error, info, or debug), and a <data-expression>.

Logging statements take only a single <data-expression> argument, so if you want to output multiple data values, you will need to use the concat operator to concatenate them.

**execute** (<command-path> [, <data-expression-1>, ..., \*<data-expression-N> ]);

The **execute** statement runs an external command. The first argument <command-path> is a string literal containing the name or path of the command to run. The other arguments, if present, are either string literals or

data expressions which evaluate to text strings, to be passed as command-line arguments to the command.

**execute** is synchronous; the process will block until the external command being run has finished.

**Warning:** Lengthy program execution (for example, in an "on commit" in *dhcpd.conf*(5)) may result in bad performance and timeouts. Only external applications with very short execution times are suitable for use.

Passing user-supplied data to an external application might be dangerous. Make sure the external application checks input buffers for validity. Non-printable ASCII characters will be converted into *dhcpd.conf*(5) language octal escapes ("`\nnn`"), make sure your external command handles them as such.

It is possible to use the **execute** statement in any context, not only on events. If it is used in a regular scope in the configuration file, that command will be executed every time a scope is evaluated.

#### **parse-vendor-option;**

The **parse-vendor-option** statement attempts to parse a vendor option (code 43). It is only useful while processing a packet on the server and requires that the administrator has already used the **vendor-option-space** statement to select a valid vendor space.

This functionality may be used if the server needs to take different actions depending on the values the client placed in the vendor option and the sub-options are not at fixed locations. It is handled as an action to allow an administrator to examine the incoming options and choose the correct vendor space.



### 3.5.7 Dynamic DNS updates

See *dhcpcd.conf(5)* and *dhclient.conf(5)* for more information about dynamic DNS updates.

### 3.5.8 See also

*dhcpcd.conf(5)*, *dhcpcd.leases(5)*, *dhclient.conf(5)*, *dhcp-options(5)*, *dhcpcd(8)*, *dhclient(8)*

### 3.5.9 Copyright

Copyright (C) 2025 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2009-2012,2014-2015 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 2004,2007 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1996-2003 by Internet Software Consortium.

## 3.6 dhcp-options --- DHCP options

### 3.6.1 Description

The Dynamic Host Configuration Protocol allows the client to receive options from the DHCP server describing the network configuration and various services that are available on the network. When configuring **dhcpcd(8)** or **dhclient(8)**, options must often be declared. The syntax for declaring options, and the names and formats of the options that can be declared, are documented here.

### 3.6.2 Option statements

**option** <option-name> <option-data>

DHCP option statements start with the **option** keyword, followed by an option name, followed by option data. The option names and data formats are described below. It is not necessary to exhaustively specify all DHCP options --- only those options which are needed by clients must be specified.

<option-data> may be specified in a variety of formats, as defined below:

The **ip-address** data type can be entered either as an explicit IP address (e.g., 239.254.197.10) or as a domain name (e.g., foo.example.com). If entering a domain name, the domain name must resolve to a single IP address.

The **ip6-address** data specifies an IPv6 address, like ::1 or 3ffe:bbbb:aaaa:aaaa::1.

The **int32** data type specifies a signed 32-bit integer. The **uint32** data type specifies an unsigned 32-bit integer. The **int16** and **uint16** data types specify signed and unsigned 16-bit integers. The **int8** and **uint8** data types specify signed and unsigned 8-bit integers. Unsigned 8-bit integers are also sometimes referred to as octets.

The **text** data type specifies an NVT-ASCII string, which must be enclosed in double quotes; for example, to specify a root-path option, the syntax would be:

```
option root-path "10.0.1.4:/var/tmp/rootfs";
```

The **domain-name** data type specifies a domain name, which must not be enclosed in double quotes. The domain name is stored just as if it were a text option.

The **domain-list** data type specifies a list of domain names, enclosed in double quotes and separated by commas (e.g., "example.com", "foo.example.com").

The **flag** data type specifies a boolean value. Booleans can be either `true` or `false`, or `on` (true) or `off` (false).

The **string** data type specifies either an NVT-ASCII string enclosed in double quotes, or a series of octets specified in hexadecimal, separated by colons. For example:

```
option dhcp-client-identifier "CLIENT-FOO";
option dhcp-client-identifier 43:4c:49:45:54:2d:46:4f:4f;
```

### 3.6.3 Setting option values using expressions

Sometimes it's helpful to be able to set the value of a DHCP option based on some value that the client has sent. To do this, you can use expression evaluation. The *dhcp-eval* (5) manual page describes how to write expressions. To assign the result of an evaluation to an option, define the option as follows:

**option** <option-name> = <expression>;

For example:

```
option hostname = binary-to-ascii (16, 8, "-",
                                   substring (hardware, 1, 6));
```

### 3.6.4 Standard DHCPv4 options

The documentation for the various options mentioned below is taken from the latest IETF draft document on DHCP options. Options not listed below may not yet be implemented, but it is possible to use such options by defining them in the configuration file. Please see the section titled *Defining new options* for more information.

Some of the options documented here are automatically generated by the DHCP server or by clients, and cannot be configured by the user. The value of such an option can be used in the configuration file of the receiving DHCP protocol agent (server or client), for example, in conditional expressions. However, the value of the option

cannot be used in the configuration file of the sending agent, because the value is determined only after the configuration file has been processed. In the following documentation, such options will be described as "not user configurable"

The standard options are:

**option all-subnets-local** *<flag>;*

This option specifies whether or not the client may assume that all subnets of the IP network to which the client is connected use the same MTU as the subnet of that network to which the client is directly connected. A value of true indicates that all subnets share the same MTU. A value of false means that the client should assume that some subnets of the directly connected network may have smaller MTUs.

**option arp-cache-timeout** *<uint32>;*

This option specifies the timeout in seconds for ARP cache entries.

**option associated-ip** *<ip-address> [, <ip-address> ... ];*

This option is part of lease query. It is used to return all of the IP addresses associated with a given DHCP client.

---

**Note:** This option is not user configurable.

---

**option bcms-controller-address** *<ip-address> [, <ip-address> ... ];*

This option configures a list of IPv4 addresses for use as Broadcast and Multicast Controller Servers ("BCMS").

**option bcms-controller-names** *<domain-list>;*

This option contains the domain names of local Broadcast and Multicast Controller Servers ("BCMS") controllers which the client may use.

**option bootfile-name** *<text>;*

This option is used to identify a bootstrap file. If supported by the client, it should have the same effect as the filename declaration. BOOTP clients are unlikely to support this option. Some DHCP clients will support it, and others actually require it.

**option boot-size** *<uint16>;*

This option specifies the length in 512-octet blocks of the default boot image for the client.

**option broadcast-address** *<ip-address>;*

This option specifies the broadcast address in use on the client's subnet. Legal values for broadcast addresses are specified in section 3.2.1.3 of STD 3 (RFC 1122).

**option capwap-ac-v4** *<ip-address> [, <ip-address> ... ];*

A list of IPv4 addresses of CAPWAP ACs that the WTP may use. The addresses are listed in preference order. This option is included based on [RFC 5417](#).

**option client-last-transaction-time** *<uint32>*;

This option is part of lease query. It allows the receiver to determine the time of the most recent access by the client. The value is a duration in seconds from when the client last communicated with the DHCP server.

---

**Note:** This option is not user configurable.

---

**option cookie-servers** *<ip-address>* [, *<ip-address>* ... ];

The cookie server option specifies a list of [RFC 865](#) cookie servers available to the client. Servers should be listed in order of preference.

**option default-ip-ttl** *<uint8>*;

This option specifies the default time-to-live that the client should use on outgoing datagrams.

**option default-tcp-ttl** *<uint8>*;

This option specifies the default TTL that the client should use when sending TCP segments. The minimum value is 1.

**option default-url** *<string>*;

The format and meaning of this option is not described in any standards document, but is claimed to be in use by Apple Computer.

**Warning:** It is not known clients may reasonably do if supplied with this option. Use at your own risk.

**option dhcp-client-identifier** *<string>*;

This option can be used to specify a DHCP client identifier in a host declaration, so that *dhcpcd(8)* can find the host record by matching against the client identifier.

Please be aware that some DHCP clients, when configured with client identifiers that are ASCII text, will prepend a zero to the ASCII text. So you may need to write:

```
option dhcp-client-identifier "\0foo";
```

rather than:

```
option dhcp-client-identifier "foo";
```

**option dhcp-lease-time** *<uint32>*;

This option is used in a client request (DHCPDISCOVER or DHCPREQUEST) to allow the client to request a lease time for the IP address. In a server reply (DHCPOFFER), a DHCP server uses this option to specify the lease time it is willing to offer.

---

**Note:** This option is not directly user configurable in the server; refer to the `max-lease-time` and `default-lease-time` server options in `dhcpd.conf(5)`.

---

**option dhcp-max-message-size** <uint16>;

This option, when sent by the client, specifies the maximum size of any response that the server sends to the client. When specified on the server, if the client did not send a `dhcp-max-message-size` option, the size specified on the server is used. This works for BOOTP as well as DHCP responses.

**option dhcp-message** <text>;

This option is used by a DHCP server to provide an error message to a DHCP client in a DHCPNAK message in the event of a failure. A client may use this option in a DHCPDECLINE message to indicate why the client declined the offered parameters.

---

**Note:** This option is not user configurable.

---

**option dhcp-message-type** <uint8>;

This option, sent by both client and server, specifies the type of DHCP message contained in the DHCP packet. Possible values (taken directly from [RFC 2132](#)) are:

1	DHCPDISCOVER
2	DHCPOFFER
3	DHCPREQUEST
4	DHCPDECLINE
5	DHCPACK
6	DHCPNAK
7	DHCPRELEASE
8	DHCPINFORM

---

**Note:** This option is not user configurable.

---

**option dhcp-option-overload** <uint8>;

This option is used to indicate that the DHCP `sname` or `file` fields are being overloaded by using them to carry DHCP options. A DHCP server

inserts this option if the returned parameters will exceed the usual space allotted for options.

If this option is present, the client interprets the specified additional fields after it concludes interpretation of the standard fields.

Legal values for this option are:

1	The <i>file</i> field is used to hold options
2	The <i>sname</i> field is used to hold options
3	Both fields are used to hold options

---

**Note:** This option is not user configurable.

---

**option dhcp-parameter-request-list** <uint8> [, <uint8> ... ];

This option, when sent by the client, specifies which options the client wishes the server to return. Normally, in *dhclient*(8), this is done using the `request` statement. If this option is not specified by the client, the DHCP server will normally return every option that is valid scope and that fits into the reply. When this option is specified on the server, the server returns the specified options. This can be used to force a client to take options that it hasn't requested, and it can also be used to tailor the response of the DHCP server for clients that may need a more limited set of options than those the server would normally return.

**option dhcp-rebinding-time** <uint32>;

This option specifies the number of seconds from the time a client gets an address until the client transitions to the REBINDING state.

This option is user configurable, but it will be ignored if the value is greater than or equal to the lease time.

To make DHCPv4+DHCPv6 migration easier in the future, any value configured in this option is also used as a DHCPv6 "T1" (renew) time.

**option dhcp-renewal-time** <uint32>;

This option specifies the number of seconds from the time a client gets an address until the client transitions to the RENEWING state.

This option is user configurable, but it will be ignored if the value is greater than or equal to the rebinding time, or lease time.

To make DHCPv4+DHCPv6 migration easier in the future, any value configured in this option is also used as a DHCPv6 "T2" (rebind) time.

**option dhcp-requested-address** <ip-address>;

This option is used by the client in a DHCPDISCOVER to request that a particular IP address be assigned.

---

**Note:** This option is not user configurable.

---

**option dhcp-server-identifier** <ip-address>;

This option is used in DHCPOFFER and DHCPREQUEST messages, and may optionally be included in the DHCPACK and DHCPNAK messages. DHCP servers include this option in the DHCPOFFER in order to allow the client to distinguish between lease offers. DHCP clients use the contents of the *server identifier* field as the destination address for any DHCP messages unicast to the DHCP server. DHCP clients also indicate which of several lease offers is being accepted by including this option in a DHCPREQUEST message.

The value of this option is the IP address of the server.

---

**Note:** This option is not directly user configurable. See the *server-identifier* server option in *dhcpcd.conf* (5).

---

**option domain-name** <text>;

This option specifies the domain name that client should use when host-names via the DNS.

**option domain-name-servers** <ip-address> [, <ip-address> ... ];

This option specifies a list of DNS (STD 13, [RFC 1035](#)) nameservers available to the client. Nameservers should be listed in order of preference.

**option domain-search** <domain-list>;

This option specifies a *search list* of domain names to be used by the client to locate not-fully-qualified domain names. The difference between this option and historic use of the **domain-name** option for the same is that this option is encoded using [RFC 1035](#) compressed labels on the wire. For example:

```
option domain-search "example.com", "sales.example.com",  
    ↪ "eng.example.com";
```

**option extensions-path** <text>;

This option specifies the name of a file containing additional options to be interpreted according to the DHCP option format as specified in [RFC 2132](#).

**option finger-server** <ip-address> [, <ip-address> ... ];

This option specifies a list of Finger servers available to the client. Servers should be listed in order of preference.

**option font-servers** <ip-address> [, <ip-address> ... ];

This option specifies a list of X Window System Font servers available to the client. Servers should be listed in order of preference.

**option geoconf-civic** *<string>*;

A string to hold the geoconf civic structure. This option is included based on [RFC 4776](#).

**option host-name** *<string>*;

This option specifies the name of the client. The name may or may not be with the local domain name (it is preferable to use the **domain-name** option to specify the domain name). See [RFC 1035](#) for character set restrictions. This option is only honored by *dhclient-script*(8) if the hostname for the client machine is not set.

**option ieee802-3-encapsulation** *<flag>*;

This option specifies whether or not the client should use Ethernet Version 2 ([RFC 894](#)) or IEEE .3 ([RFC 1042](#)) encapsulation if the interface is of Ethernet type. A value of false indicates that the client should use [RFC 894](#) encapsulation. A value of true means that the client should use [RFC 1042](#) encapsulation.

**option ien116-name-servers** *<ip-address>* [, *<ip-address>* ... ];

This option specifies a list of IEN 116 name servers available to the client. Servers should be listed in order of preference.

**option impress-servers** *<ip-address>* [, *<ip-address>* ... ];

This option specifies a list of Imagen Impress servers available to the client. Servers should be listed in order of preference.

**option interface-mtu** *<uint16>*;

This option specifies the MTU to use on this interface. The minimum legal value for the MTU is 68.

**option ip-forwarding** *<flag>*;

This option specifies whether the client should configure its IP layer for packet forwarding. A value of false means disable IP forwarding, and a value of true means enable IP forwarding.

**option irc-server** *<ip-address>* [, *<ip-address>* ... ];

This option specifies a list of IRC servers available to the client. Servers should be listed in order of preference.

**option log-servers** *<ip-address>* [, *<ip-address>* ... ];

This option specifies a list of MIT-LCS UDP log servers available to the client. Servers should be listed in order of preference.

**option lpr-servers** *<ip-address>* [, *<ip-address>* ... ];



This option specifies a list of [RFC 1179](#) line printer servers available to the client. Servers should be listed in order of preference.

**option mask-supplier** <flag>;

This option specifies whether or not the client should respond to subnet mask requests using ICMP. A value of false indicates that the client should not respond. A value of true means that the client should respond.

**option max-dgram-reassembly** <uint16>;

This option specifies the maximum size datagram that the client should be prepared to reassemble. The minimum legal value is 576.

**option merit-dump** <text>;

This option specifies the path-name of a file to which the client's core image should be dumped in the event the client crashes. The path is formatted as a character string consisting of characters from the NVT-ASCII character set.

**option mobile-ip-home-agent** <ip-address> [, <ip-address> ... ];

This option specifies a list of IP addresses indicating mobile IP home agents available to the client. Agents should be listed in order of preference, although normally there will be only one such agent.

**option name-service-search** <uint16> [, <uint6> ... ];

This option specifies a list of name services in the order the client should attempt to use them. This option is included based on [RFC 2937](#).

**option nds-context** <string>;

This option specifies the name of the initial Netware Directory Service for an NDS client.

**option nds-servers** <ip-address> [, <ip-address> ... ];

This option specifies a list of IP addresses of NDS servers.

**option nds-tree-name** <string>;

This option specifies the NDS tree name that the NDS client should use.

**option netbios-dd-server** <ip-address> [, <ip-address> ... ];

This option specifies a list of [RFC 1001](#) and [RFC 1002](#) NetBIOS datagram distribution server (NBDD) servers listed in order of preference.

**option netbios-name-servers** <ip-address> [, <ip-address> ... ];

This option specifies a list of [RFC 1001](#) and [RFC 1002](#) NetBIOS name servers (NBNS) listed in order of preference. NetBIOS Name Service is currently more commonly referred to as WINS. WINS servers can be specified using this option.

**option netbios-node-type** <uint8>;

This option allows NetBIOS over TCP/IP clients (which are configurable) to be configured as described in [RFC 1001](#) and [RFC 1002](#). The NetBIOS node type value is specified as a single octet which identifies the client type.

Possible node types are:

1	B-node: Broadcast - no WINS
2	P-node: Peer - WINS only
4	M-node: Mixed - broadcast, then WINS
8	H-node: Hybrid - WINS, then broadcast

**option netbios-scope** *<string>*;

This option specifies the NetBIOS over TCP/IP scope for the client as specified in [RFC 1001](#) and [RFC 1002](#). See [RFC 1001](#), [RFC 1002](#), and [RFC 1035](#) for character-set restrictions.

**option netinfo-server-address** *<ip-address>* [, *<ip-address>* ... ];

The format and meaning of this option is not described in any standards document, but is claimed to be in use by Apple Computer.

**Warning:** It is not known clients may reasonably do if supplied with this option. Use at your own risk.

**option netinfo-server-tag** *<text>*;

The format and meaning of this option is not described in any standards document, but is claimed to be in use by Apple Computer.

**Warning:** It is not known clients may reasonably do if supplied with this option. Use at your own risk.

**option nis-domain** *<text>*;

This option specifies the name of the client's NIS (Sun Network Information Services) domain. The domain is formatted as a character string consisting of characters from the NVT-ASCII character set.

**option nis-servers** *<ip-address>* [, *<ip-address>* ... ];

This option specifies a list of IP addresses indicating NIS servers available to the client. Servers should be listed in order of preference.

**option nisplus-domain** *<text>*;

This option specifies the name of the client's NIS+ domain. The domain is formatted as a character string consisting of characters from the NVT-ASCII character set.

**option nisplus-servers** <ip-address> [, <ip-address> ... ];

This option specifies a list of IP addresses indicating NIS+ servers available to the client. Servers should be listed in order of preference.

**option nnntp-server** <ip-address> [, <ip-address> ... ];

The NNTP server option specifies a list of NNTP servers available to the client. Servers should be listed in order of preference.

**option non-local-source-routing** <flag>;

This option specifies whether the client should configure its IP layer to allow forwarding of datagrams with non-local source routes. See Section 3.3.5 of STD 3 ([RFC 1122](#)) for a discussion of this topic. A value of false means disallow forwarding of such datagrams, and a value of true means allow forwarding.

**option ntp-servers** <ip-address> [, <ip-address> ... ];

This option specifies a list of IP addresses indicating NTP ([RFC 5905](#)) servers available to the client. Servers should be listed in order of preference.

**option nwip-domain** <string>;

The name of the NetWare/IP domain that a NetWare/IP client should use.

**option nwip-suboptions** <string>;

A sequence of suboptions for NetWare/IP clients --- see [RFC 2242](#) for details. Normally this is set by specifying specific NetWare/IP suboptions. See the section titled *NetWare/IP suboptions* for more information.

**option option-6rd** <uint8> <uint8> <ip6-address> <ip-address> [, <ip-address> ... ];

This option contains information about the rapid deployment option. It is 8 bits of IPv4 mask length, 8 bits of 6rd prefix length, an IPv6 prefix as an IPv6 address and a list of one or more IPv4 addresses. This option is included based on [RFC 5969](#).

**option pana-agent** <ip-address> [, <ip-address> ... ];

A set of IPv4 addresses of a PAA for the client to use. The addresses are listed in preferred order. This option is included based on [RFC 5192](#).

**option path-mtu-aging-timeout** <uint32>;

This option specifies the timeout (in seconds) to use when aging Path MTU values discovered by the mechanism defined in [RFC 1191](#).

**option path-mtu-plateau-table** <uint16> [, <uint16> ... ];

This option specifies a table of MTU sizes to use when performing Path MTU Discovery as defined in [RFC 1191](#). The table is formatted as a list of 16-bit unsigned integers, ordered from smallest to largest. The minimum MTU value cannot be smaller than 68.

**option pcode** <text>;

This option specifies a string suitable for the TZ variable. This option is included based on [RFC 4833](#).

**option perform-mask-discovery** <flag>;

This option specifies whether or not the client should perform subnet mask discovery using ICMP. A value of false indicates that the client should not perform mask discovery. A value of true means that the client should perform mask discovery.

**option policy-filter** ip-address ip-address [, ip-address ip-address...];

This option specifies policy filters for non-local source routing. The filters consist of a list of IP addresses and masks which specify destination/mask pairs with which to filter incoming source routes.

Any source routed datagram whose next-hop address does not match one of the filters should be discarded by the client.

See STD 3 ([RFC 1122](#)) for further information.

**option pop-server** <ip-address> [, <ip-address> ... ];

This option specifies a list of POP3 servers available to the client. Servers should be listed in order of preference.

**option rdns-selection** <uint8> <ip-address> <ip-address> <domain-list>;

This option specifies an 8-bit flags field, a primary and secondary IP address for the nameserver, and a <domain-list> of domains for which the RDNS has special knowledge. This option is included based on [RFC 6731](#).

**option resource-location-servers** <ip-address> [, <ip-address> ... ];

This option specifies a list of [RFC 887](#) Resource Location servers available to the client. Servers should be listed in order of preference.

**option root-path** <text>;

This option specifies the path-name that contains the client's root disk. The path is formatted as a character string consisting of characters from the NVT-ASCII character set.

**option router-discovery** <flag>;

This option specifies whether or not the client should solicit routers using the Router Discovery mechanism defined in [RFC 1256](#). A value of false that the client should not perform router discovery. A value of true means that the client should perform router discovery.

**option router-solicitation-address** <ip-address>;

This option specifies the address to which the client should transmit router solicitation requests.

**option routers** <ip-address> [, <ip-address> ... ];

The routers option specifies a list of IP addresses for routers on the client's subnet. Routers should be listed in order of preference.

**option slp-directory-agent** <flag> <ip-address> [, <ip-address> ... ];

This option specifies two things: the IP addresses of one or more Service Location Protocol Directory Agents, and whether the use of these addresses is mandatory. If the initial flag value is true, the SLP agent should just use the IP addresses given. If the value is false, the SLP agent may additionally do active or passive multicast discovery of SLP agents (see [RFC 2165](#) for details).

---

**Note:** In this option and the **slp-service-scope** option, the term "SLP Agent" is being used to refer to a Service Location Protocol agent running on a machine that is being configured using the DHCP protocol.

---

---

**Note:** Please be aware that some companies may refer to SLP as NDS. If you have an NDS directory agent whose address you need to configure, this option should work.

---

**option slp-service-scope** <flag> <text>;

The Service Location Protocol Service Scope Option specifies two things: a list of service scopes for SLP, and whether the use of this list is mandatory. If the initial flag value is true, the SLP agent should only use the list of scopes provided in this option; otherwise, it may use its own static configuration in preference to the list provided in this option.

The text string should be a comma-separated list of scopes that the SLP agent should use. It may be omitted, in which case the SLP Agent will use the aggregated list of scopes of all directory agents known to the SLP agent.

**option smtp-server** <ip-address> [, <ip-address> ... ];

The SMTP server option specifies a list of SMTP servers available to the client. Servers should be listed in order of preference.

**option static-routes** <ip-address> <ip-address> [, <ip-address> <ip-address> ... ];

This option specifies a list of static routes that the client should install in its routing cache. If multiple routes to the same destination are specified, they are listed in descending order of priority.

The routes consist of a list of IP address pairs. The first address is the destination address, and the second address is the router for the destination.

The default route (0.0.0.0) is an illegal destination for a static route. To specify the default route, use the **routers** option. Also, please note that this option is not intended for classless IP routing --- it does not include a subnet mask. Since classless IP routing is now the most widely deployed routing

standard, this option is virtually useless, and is not implemented by any of the popular DHCP clients, for example the Microsoft DHCP client.

**option streettalk-directory-assistance-server** *<ip-address>* [, *<ip-address>* ... ];

This option specifies a list of StreetTalk Directory Assistance (STDA) servers available to the client. Servers should be listed in order of preference.

**option streettalk-server** *<ip-address>* [, *<ip-address>* ... ];

This option specifies a list of StreetTalk servers available to the client. Servers should be listed in order of preference.

**option subnet-mask** *<ip-address>*;

This option specifies the client's subnet mask as per [RFC 950](#). If no subnet mask option is provided anywhere in scope, as a last resort *dhcpcd(8)* will use the subnet mask from the subnet declaration for the network on which an address is being assigned. However, any **subnet-mask** option declaration that is in scope for the address being assigned will override the subnet mask specified in the subnet declaration.

**option subnet-selection** *<ip-address>*;

Sent by the client if an address is required in a subnet other than the one that would normally be selected (based on the relaying address of the connected subnet the request is obtained from). See [RFC 3011](#).

---

**Note:** The option number used by this server is 118; this has not always been the defined number, and some clients may use a different value.

---

**Warning:** Use of this option should be regarded as slightly experimental.

---

**Note:** This option is not user configurable in the server.

---

**option swap-server** *<ip-address>*;

This specifies the IP address of the client's swap server.

**option tcp-keepalive-garbage** *<flag>*;

This option specifies whether or not the client should send TCP keepalive messages with an octet of garbage for compatibility with older implementations. A value of false indicates that a garbage octet should not be sent. A value of true indicates that a garbage octet should be sent.

**option tcp-keepalive-interval** *<uint32>*;

This option specifies the interval (in seconds) that the client TCP should wait before sending a keepalive message on a TCP connection. The time is specified as a 32-bit unsigned integer. A value of 0 indicates that the client should not generate keepalive messages on connections unless specifically requested by an application.

**option tcode** *<text>*;

This option specifies a name of a zone entry in the TZ database. This option is included based on [RFC 4833](#).

**option tftp-server-name** *<text>*;

This option is used to identify a TFTP server and, if supported by the client, should have the same effect as the **server-name** declaration. BOOTP clients are unlikely to support this option. Some DHCP clients will support it, and others actually require it.

**option time-offset** *<int32>*;

This option specifies the offset of the client's subnet in seconds from Coordinated Universal Time (UTC).

**option time-servers** *<ip-address> [, <ip-address> ... ]*;

This option specifies a list of [RFC 868](#) time servers available to the client. Servers should be listed in order of preference.

**option trailer-encapsulation** *<flag>*;

This option specifies whether or not the client should negotiate the use of trailers ([RFC 893](#)) when using the ARP protocol. A value of false indicates that the client should not attempt to use trailers. A value of true means that the client should attempt to use trailers.

**option uap-servers** *<text>*;

This option specifies a list of URLs, each pointing to a user authentication service that is capable of processing authentication requests encapsulated in the User Authentication Protocol (UAP). UAP servers can accept either HTTP/1.1 or SSLv3 connections. If the list includes a URL that does not contain a port component, the normal default port is assumed (i.e., port 80 for HTTP and port 443 for HTTPS). If the list includes a URL that does not contain a path component, the path `/uap` is assumed. If more than one URL is specified in this list, the URLs are separated by spaces.

**option user-class** *<string>*;

This option is used by some DHCP clients as a way for users to specify identifying information to the client. This can be used in a similar way to the **vendor-class-identifier** option, but the value of the option is specified by the user, not the vendor. Most recent DHCP clients have a way in the user interface to specify the value for this identifier, usually as a text string.

**option v4-access-domain** *<domain-name>*;



The domain name associated with the access network for use with LIS Discovery. This option is included based on [RFC 5986](#).

**option v4-lost** <domain-name>;

The domain name of the LoST server for the client to use. This option is included based on [RFC 5223](#).

**option vendor-class-identifier** <string>;

This option is used by some DHCP clients to identify the vendor type and possibly the configuration of a DHCP client. The information is a string of bytes whose contents are specific to the vendor and are not specified in a standard. To see what vendor class identifier clients are sending, you can write the following in your DHCP server configuration file:

```
set vendor-string = option vendor-class-identifier;
```

This will result in all entries in the DHCP server lease database file for clients that sent **vendor-class-identifier** options having a set statement that looks something like this:

```
set vendor-string = "SUNW.Ultra-5_10";
```

The **vendor-class-identifier** option is normally used by the DHCP server to determine the options that are returned in the **vendor-encapsulated-options** option. Please see the section titled *Vendor encapsulated options* for further information.

**option vendor-encapsulated-options** <string>;

This option can contain either a single vendor-specific value or one or more vendor-specific suboptions. This option is not normally specified in the DHCP server configuration file --- instead, a vendor class is defined for each vendor, vendor class suboptions are defined, values for those suboptions are defined, and the DHCP server makes up a response on that basis.

Some default behaviours for well-known DHCP client vendors (currently, the Microsoft Windows 2000 DHCP client) are configured automatically, but otherwise this must be configured manually. Please see the section titled *Vendor encapsulated options* for further information.

**option vivso** <string>;

This option can contain multiple separate options, one for each 32-bit Enterprise ID. Each Enterprise-ID discriminated option then contains additional options whose format is defined by the vendor who holds that ID. This option is usually not configured manually, but rather is configured via intervening option definitions. Please see the section titled *Vendor encapsulated options* for further information.

**option www-server** <ip-address> [, <ip-address> ... ];



The WWW server option specifies a list of WWW servers available to the client. Servers should be listed in order of preference.

**option x-display-manager** <ip-address> [, <ip-address> ... ];

This option specifies a list of systems that are running the X Window System Display Manager and are available to the client. Addresses should be listed in order of preference.

### 3.6.5 Relay agent information option

**RFC 3046** defines a series of encapsulated options that a relay agent can add to a DHCP packet when relaying it to the DHCP server. The server can then make address allocation decisions (or whatever other decisions it wants) based on these options. The server also returns these options in any replies it sends through the relay agent, so that the relay agent can use the information in these options for delivery or accounting purposes.

**RFC 3046** defines two options. To reference these options in the DHCP server, specify the option space name, "agent", followed by a period, followed by the option name. It is not normally useful to define values for these options in the server, although it is permissible. These options are not supported in the client.

**option agent.circuit-id** <string>;

The **circuit-id** suboption encodes an agent-local identifier of the circuit from which a DHCP client-to-server packet was received. It is intended for use by agents in relaying DHCP responses back to the proper circuit. The format of this option is currently defined to be vendor-dependent, and will probably remain that way, although the current draft allows for the possibility of standardizing the format in the future.

**option agent.remote-id** <string>;

The **remote-id** suboption encodes information about the remote host end of a circuit. Examples of what it might contain include caller ID information, username information, remote ATM address, cable modem ID, and similar things. In principal, the meaning is not well-specified, and it should generally be assumed to be an opaque object that is administratively guaranteed to be unique to a particular remote end of a circuit.

**option agent.DOCSIS-device-class** <uint32>;

The **DOCSIS-device-class** suboption is intended to convey information about the host endpoint, hardware, and software, that either the host operating system or the DHCP server may not otherwise be aware of (but the relay is able to distinguish). This is implemented as a 32-bit field (4 octets), each bit representing a flag describing the host in one of these ways. So far, only bit zero (being the least significant bit) is defined in **RFC 3256**. If this bit is set to 1, the host is considered a CPE Controlled Cable Modem (CCCM). All other bits are reserved.

**option agent.link-selection** <ip-address>;

The **link-selection** suboption is provided by relay agents to inform servers what subnet the client is actually attached to. This is useful in those cases where the *giaddr* (where responses must be sent to the relay agent) is not on the same subnet as the client. When this option is present in a packet from a relay agent, the DHCP server will use its contents to find a subnet declared in configuration, and from here take one step further backwards to any shared-network the subnet may be defined within; the client may be given any address within that shared network, as normally appropriate.

### 3.6.6 Client FQDN suboptions

The Client FQDN option is defined in [RFC 4702](#). Due to the complexity of the option format, it is implemented as a suboption space rather than a single option. In general this option should not be configured by the user --- instead it should be used as part of an automatic DNS UPDATE system.

---

**Note:** If you wish to use any of these suboptions, we strongly recommend that refer to [RFC 4702](#). The documentation here is sketchy and incomplete in comparison, and is just intended for reference by people who already understand the Client FQDN option specification.

---

**option fqdn.no-client-update** <flag>;

When the client sends this option, if it is true, it means the client will not attempt to update its A record. When sent by the server to the client, it means that the client should not update its own A record.

**option fqdn.server-update** <flag>;

When the client sends this to the server, it is requesting that the server update its A record. When sent by the server, it means that the has updated (or is about to update) the client's A record.

**option fqdn.encoded** <flag>;

If true, this indicates that the domain name included in the option is encoded in DNS wire format, rather than as plain ASCII text. The client normally sets this to false if it doesn't support DNS wire format in the FQDN option. The server should always send back the same value that the client sent. When this value is set on the configuration side, it controls the format in which the **fqdn.fqdn** suboption is encoded.

**option fqdn.rcode1** <flag>;

**option fqdn.rcode2** <flag>;

These options specify the result of the updates of the A and PTR records, respectively, and are only sent by the DHCP server to the DHCP client. The values of these fields are those defined in the DNS protocol specification.

**option fqdn.fqdn** *<text>*;

Specifies the domain name that the client wishes to use. This can be a fully-qualified domain name, or a single label. If there is no trailing `.` character in the name, it is not fully-qualified, and the server will generally update that name in some locally-defined domain.

**option fqdn.hostname** [never-set];

This option should never be set, but it can be read back using the **option** and **config-option** operators in an expression, in which case it returns the first label in the **fqdn.fqdn** suboption. For example, if the value of **fqdn.fqdn** is `"foo.example.com."`, then **fqdn.hostname** will be `"foo"`.

**option fqdn.domainname** [never-set];

This option should never be set, but it can be read back using the **option** and **config-option** operators in an expression, in which case it returns all labels after the first label in the **fqdn.fqdn** suboption. For example, if the value of **fqdn.fqdn** is `"foo.example.com."`, then **fqdn.domainname** will be `"example.com."`. If this suboption value is not set, it means that an unqualified name was sent in the Client FQDN option, or that no Client FQDN option was sent at all.

### 3.6.7 NetWare/IP suboptions

**RFC 2242** defines a set of encapsulated options for NetWare/IP clients. To use these options in the DHCP server, specify the option space name `"nwip"` followed by a period followed by the option name. The following options can be specified:

**option nwip.nsq-broadcast** *<flag>*;

If true, the client should use the NetWare Nearest Server Query to locate a NetWare/IP server. The behaviour of the Novell client if this suboption is false, or is not present, is not specified.

**option nwip.preferred-dss** *<ip-address>* [, *<ip-address>* ... ];

This suboption specifies a list of up to five IP addresses, each of which should be the IP address of a NetWare Domain SAP/RIP server (DSS).

**option nwip.nearest-nwip-server** *ip-address* [, *ip-address...*];

This suboption specifies a list of up to five IP addresses, each of which should be the IP address of a Nearest NetWare IP server.

**option nwip.autoretries** *<uint8>*;

Specifies the number of times that a NetWare/IP client should attempt to communicate with a given DSS server at startup.

**option nwip.retry-secs** *<uint8>*;

Specifies the number of seconds that a Netware/IP client should wait between retries when attempting to establish communications with a DSS server at startup.

**option nwip.nwip-1-1** <uint8>;

If true, the NetWare/IP client should support NetWare/IP version 1.1 compatibility. This is only needed if the client will be contacting Netware/IP version 1.1 servers.

**option nwip.primary-dss** ip-addressfB;

Specifies the IP address of the Primary Domain SAP/RIP Service server (DSS) for this NetWare/IP domain. The NetWare/IP administration utility uses this value as Primary DSS server when configuring a secondary DSS server.

### 3.6.8 Standard DHCPv6 options

DHCPv6 options differ from DHCPv4 options partially due to using 16-bit code and length tags, but semantically zero-length options are legal in DHCPv6, and multiple options are treated differently. Whereas in DHCPv4 multiple options would be concatenated to form one option, in DHCPv6 they are expected to be individual instantiations. Understandably, many options are not "allowed" to have multiple instances in a packet - normally these are options which are digested by the DHCP protocol software, and not by users or applications.

**option dhcp6.client-id** <string>;

This option specifies the client's DUID identifier. DUIDs are similar but different from DHCPv4 client identifiers --- there are documented DUID types:

- duid-llt
- duid-en
- duid-ll

This value should not be configured, but rather is provided by clients and treated as an opaque identifier key blob by servers.

**option dhcp6.server-id** <string>;

This option specifies the server's DUID identifier. This option may be used to configure an opaque binary blob for a DHCP server's identifier.

**option dhcp6.ia-na** <string>;

The Identity Association for Non-temporary Addresses (**ia-na**) carries assigned addresses that are not temporary addresses for use by the DHCPv6 client. This option is produced by the DHCPv6 server software, and should not be configured.

**option dhcp6.ia-ta** <string>;

The Identity Association for Temporary Addresses (**ia-ta**) carries temporary addresses, which may change upon every renewal. There is no support for this in the current DHCPv6 software.

**option dhcp6.ia-addr** <string>;

The Identity Association Address option is encapsulated inside **ia-na** or **ia-ta** options in order to represent addresses associated with those IAs. These options are manufactured by the software, so should not be configured.

**option dhcp6.oro** <uint16> [, <uint16> ... ];

The Option Request Option ("ORO") is the DHCPv6 equivalent of the **parameter-request-list** option. Clients supply this option to ask servers to reply with options relevant to their needs and use. This option must not be directly configured, the request syntax in *dhclient.conf*(5) should be used instead.

**option dhcp6.preference** <uint8>;

The **preference** option informs a DHCPv6 client which server is *preferred* for use on a given subnet. This preference is only applied during the initial stages of configuration --- once a client is bound to an IA, it will remain bound to that IA until it is no longer valid or has expired. This value may be configured on the server, and is digested by the client software.

**option dhcp6.elapsed-time** <uint16>;

The **elapsed-time** option is constructed by the DHCPv6 client software, and is potentially consumed by intermediaries. This option should not be configured.

**option dhcp6.relay-msg** <string>;

The **relay-msg** option is constructed by intervening DHCPv6 relay agent software. This option is entirely used by protocol software, and is not meant for user configuration.

**option dhcp6.unicast** <ip6-address>;

The **unicast** option is provided by DHCPv6 servers which are willing (or prefer) to receive Request, Renew, Decline, and Release packets from their clients via unicast. Normally, DHCPv6 clients will multicast these messages. Per [RFC 3315](#), the server will reject a unicast message received from a client unless it previously sent (or would have sent) the unicast option to that client. This option may be configured on the server at the global and shared network level. When a unicast message is received, the server will check an applicable definition of the unicast option. If such an option is found, the message will be accepted; if not it will be rejected.

**option dhcp6.status-code** <status-code> [ <string> ];

The **status-code** option is provided by DHCPv6 servers to inform clients of error conditions during protocol communication. This option is manufactured and digested by protocol software, and should not be configured.

**option dhcp6.rapid-commit ;**

The **rapid-commit** option is a zero-length option that clients use to indicate their desire to enter into rapid-commit with the server.

**option dhcp6.vendor-opts <string>;**

The **vendor-opts** option is actually an encapsulated sub-option space, in which each Vendor-specific Information Option (VSIO) is identified by a 32-bit Enterprise-ID number. The encapsulated option spaces within these options are defined by the vendors.

To make use of this option, the best way is to examine the section titled *Vendor encapsulated options*, in particular the bits about the "vsio" option space.

**option dhcp6.interface-id <string>;**

The **interface-id** option is manufactured by relay agents, and may be used to guide configuration differentiating clients by the interface they are remotely attached to. It does not make sense to configure a value for this option, but it may make sense to inspect its contents.

**option dhcp6.reconf-msg <dhcpv6-message>;**

The **reconf-msg** option is manufactured by servers, and sent to clients in Reconfigure messages to inform them of what message the client should Reconfigure using. There is no support for DHCPv6 Reconfigure extensions, and this option is documented informationally only.

**option dhcp6.reconf-accept ;**

The **reconf-accept** option is a zero-length option that is included by DHCPv6 clients that support the Reconfigure extensions, advertising that they will respond if the server to ask them to Reconfigure. There is no support for DHCPv6 Reconfigure extensions, and this option is documented informationally.

**option dhcp6.sip-servers-names <domain-list>;**

The **sip-servers-names** option allows SIP clients to locate a local SIP server that is to be used for all outbound SIP requests, a so-called "outbound proxy server." If IPv6 addresses should be supplied instead, the **sip-servers-addresses** option may be used.

**option dhcp6.sip-servers-addresses <ip6-address> [, <ip6-address> ... ];**

The **sip-servers-addresses** option allows SIP clients to locate a local SIP server that is to be used for all outbound SIP requests, a so-called "outbound proxy servers." If domain names should be supplied instead, the **sip-servers-names** option may be used.

**option dhcp6.name-servers <ip6-address> [, <ip6-address> ... ];**

The **name-servers** option instructs clients about locally available recursive DNS servers. It is easiest to describe this as the `nameserver` statement in

`/etc/resolv.conf`.

**option dhcp6.domain-search** *<domain-list>*;

The **domain-search** option specifies the client's domain search path to be applied to recursive DNS queries. It is easiest to describe this as the search statement in `/etc/resolv.conf`.

**option dhcp6.ia-pd** *<string>*;

The **ia-pd** option is manufactured by clients and servers to create a Prefix Delegation binding --- to delegate an IPv6 prefix to the client. It is not directly edited in `dhcpcd.conf(5)` or `dhclient.conf(5)`, but rather is manufactured and consumed by the software.

**option dhcp6.ia-prefix** *<string>*;

The **ia-prefix** option is placed inside *ia-pd* options in order to identify the prefix(es) allocated to the client. It is not directly edited in `dhcpcd.conf(5)` or `dhclient.conf(5)`, but rather is manufactured and consumed by the software.

**option dhcp6.nis-servers** *<ip6-address>* [, *<ip6-address>* ... ];

The **nis-servers** option identifies, in order, NIS servers available to the client.

**option dhcp6.nisp-servers** *<ip6-address>* [, *<ip6-address>* ... ];

The **nisp-servers** option identifies, in order, NIS+ servers available to the client.

**option nis-domain-name** *<domain-list>*;

The **nis-domain-name** option specifies the NIS domain name the client is expected to use, and is related to the **nis-servers** option.

**option dhcp6.nis-domain-name** *<domain-name>*;

The **nis-domain-name** option specifies NIS domain name the client is expected to use, and is related to **nis-servers** option.

**option nisp-domain-name** *<domain-list>*;

The **nisp-domain-name** option specifies the NIS+ domain name the client is expected to use, and is related to the **nisp-servers** option.

**option dhcp6.nisp-domain-name** *<domain-name>*;

The **nisp-domain-name** option specifies NIS+ domain name the client is expected to use, and is related to **nisp-servers** option.

**option dhcp6.sntp-servers** *<ip6-address>* [, *<ip6-address>* ... ];

The **sntp-servers** option specifies a list of local SNTP servers available for the client to synchronize their clocks.

**option dhcp6.info-refresh-time** *<uint32>*;



The **info-refresh-time** option gives DHCPv6 clients using Information-request messages a hint as to how long they should wait between refreshing the information they were given.

---

**Note:** This option will only be delivered to the client, and be likely to affect the client's behaviour, if the client requested the option.

---

**option dhcp6.bcms-server-d** <domain-list>;

The **bcms-server-d** option contains the domain names of local BCMS (Broadcast and Multicast Control Services) controllers which the client may use.

**option dhcp6.bcms-server-a** <ip6-address> [, <ip6-address> ... ];

The **bcms-server-a** option contains the IPv6 addresses of local BCMS (Broadcast and Multicast Control Services) controllers which the client may use.

**option dhcp6.geoconf-civic** <string>;

A string to hold the geoconf civic structure. This option is included based on [RFC 4776](#).

**option dhcp6.remote-id** <string>;

The **remote-id** option is constructed by relay agents, to inform the server of details pertaining to what the relay knows about the client (as what port it is attached to, and so forth). The contents of this option have some vendor-specific structure (similar to VSIO), but we have chosen to treat this option as an opaque field.

**option dhcp6.subscriber-id** <string>;

The **subscriber-id** option is an opaque field provided by the relay agent, which provides additional information about the subscriber in question. The exact contents of this option depend upon the vendor and/or the operator's configuration of the remote device, and as such is an opaque field.

**option dhcp6.fqdn** <string>;

The **fqdn** option is normally constructed by the client or server, and negotiates the client's Fully Qualified Domain Name, as well as which is responsible for Dynamic DNS Updates. See the section titled [Client FQDN suboptions](#) for full details (the DHCPv4 and DHCPv6 FQDN options use the same "fqdn." encapsulated space, so are in all ways identical).

**option dhcp6.pana-agent** <ip6-address> [, <ip6-address> ... ];

A set of IPv6 addresses of a PAA for the client to use. The addresses are listed in preferred order. This option is included based on [RFC 5192](#).

**option dhcp6.new-posix-timezone** <text>;



This option specifies a string suitable for the TZ variable. This option is included based on [RFC 4833](#).

**option dhcp6.new-tzdb-timezone** <text>;

This option specifies a name of a zone entry in the TZ database. This option is included based on [RFC 4833](#).

**option dhcp6.ero** <uint16> [, <uint16> ... ];

A list of the options requested by the relay agent. This option is included based on [RFC 4994](#).

**option dhcp6.lq-query** <string>;

The **lq-query** option is used internally for lease query.

**option dhcp6.client-data** <string>;

The **client-data** option is used internally for lease query.

**option dhcp6.clt-time** <uint32>;

The **clt-time** option is used internally for lease query.

**option dhcp6.lq-relay-data** <ip6-address> <string>;

The **lq-relay-data** option is used internally for lease query.

**option dhcp6.lq-client-link** <ip6-address> [, <ip6-address> ... ];

The **lq-client-link** option is used internally for lease query.

**option dhcp6.v6-lost** <domain-name>;

The domain name of the LoST server for the client to use. This option is included based on [RFC 5223](#).

**option dhcp6.capwap-ac-v6** <ip6-address> [, <ip6-address> ... ];

A list of IPv6 addresses of CAPWAP ACs that the WTP may use. The addresses are listed in preference order. This option is included based on [RFC 5417](#).

**option dhcp6.relay-id** <string>;

The DUID for the relay agent. This option is included based on [RFC 5460](#).

**option dhcp6.v6-access-domain** <domain-name>;

The domain name associated with the access network for use with LIS Discovery. This option is included based on [RFC 5986](#).

**option dhcp6.sip-ua-cs-list** <domain-list>;

The list of domain names in the SIP User Agent Configuration Service Domains. This option is included based on [RFC 6011](#).

**option dhcp6.bootfile-url** <text>;

The URL for a boot file. This option is included based on [RFC 5970](#).

**option dhcp6.bootfile-param** *<string>;*

A string for the parameters to the bootfile. See [RFC 5970](#) for more description of the layout of the parameters within the string. This option is included based on [RFC 5970](#).

**option dhcp6.client-arch-type** *<uint16> [, <uint16> ... ];*

A list of one or more architecture types described as 16-bit values. This option is included based on [RFC 5970](#).

**option dhcp6.nii** *<uint8> <uint8> <uint8>;*

The **nii** (client network interface identifier) option supplies information about a client's level of UNDI support. The values are, in order, the type, the major value and the minor value. This option is included based on [RFC 5970](#).

**option dhcp6.aftr-name** *<domain-name>;*

A domain name of the AFTR tunnel endpoint. This option is included based on [RFC 6334](#).

**option dhcp6.erp-local-domain-name** *<domain-name>;*

A domain name for the ERP domain. This option is included based on [RFC 6440](#).

**option dhcp6.rdnss-selection** *<ip6-address> <uint8> <domain-name>;*

RDNSS information consists of an IPv6 address of RDNSS, an 8-bit flags field and a domain-list of domains for which the RDNSS has special knowledge. This option is included based on [RFC 6731](#).

**option dhcp6.client-linklayer-addr** *<string>;*

A client link-layer address. The first two bytes must be the type of the link-layer followed by the address itself. This option is included based on [RFC 6939](#).

**option dhcp6.link-address** *<ip6-address>;*

An IPv6 address used by a relay agent to indicate to the server the link on which the client is located. This option is included based on [RFC 6977](#).

**option dhcp6.solmax-rt** *<uint32>;*

A value to override the default for SOL\_MAX\_RT. This is a 32-bit value. This option is included based on [RFC 7083](#).

**option dhcp6.inf-max-rt** *<uint32>;*

A value to override the default for INF\_MAX\_RT. This is a 32-bit value. This option is included based on [RFC 7083](#).

### 3.6.9 Accessing DHCPv6 relay options

**v6relay** (<relay-number>, \*<option>)

This function allows access to an option that has been added to a packet by a relay agent. Relay-number value selects the relay to examine and option is the option to find. In DHCPv6 each relay encapsulates the entire previous message into an option, adds its own options (if any) and sends the result onwards. The RFC specifies a limit of 32 hops. A relay-number of 0 is a no-op and means don't look at the relays. 1 is the relay that is closest to the client, 2 would be the next in from the client and so on. Any value greater than the max number of hops is which is closest to the server independent of number. To use this option in a class statement, a statement similar to the following may be used:

```
match if v6relay(1, option dhcp6.subscriber-id) = "client_1
→";
```

### 3.6.10 Defining new options

*dhclient(8)* and *dhcpcd(8)* provide the capability to define new options. Each DHCP option has a name, a code, and a structure. The name can be used by a human to refer to the option. The code is a number, used by the DHCP server and client to refer to an option. The structure describes what the content of an option looks like.

To define a new option, a name must be chosen for it that is not in use for some other option. For example, the "host-name" option name cannot be used because the DHCP protocol already defines a **host-name** option, which is documented in this manual page. If an option name doesn't appear in this manual page, it can be used, but it's probably a good idea to put some kind of unique string prefix at the beginning of the name so that future options don't clash with this name. For example, an option may be named as "my-local-host-name" with some confidence that no official DHCP option name will ever begin with the "my-local-" string prefix.

Once a name has been chosen for the option, a code has to be chosen for it next. All codes between 224 and 254 are reserved as site-local DHCP options, so any one of these values may be used for a local site (but not for a product/application). In [RFC 3942](#), site-local space was moved from starting at 128 to starting at 224. In practice, some vendors have interpreted the protocol rather loosely and have used option code values greater than 128 themselves. There's no real way to avoid this problem, and it was thought to be unlikely to cause too much trouble in practice. If a vendor-documented option code uses values in either the new or old site-local spaces, please contact the vendor and inform them about [RFC 3942](#).

The structure of an option is simply the format in which the option data appears. *dhcpcd(8)* currently supports a few simple types, like integers, booleans, strings and IP addresses, and it also supports the ability to define arrays of single types or arrays of fixed sequences of types.

New options are declared as follows:

**option** <new-name> **code** <new-code> = <definition>;

The values of <new-name> and <new-code> should be the chosen name and code for the new option. The definition should be the definition of the structure of the option.

The following simple option type definitions are supported:

### 3.6.10.1 Boolean

**option** <new-name> **code** <new-code> = **boolean**;

An option of type **boolean** is a flag with a value of either `true` or `false`, or `on` (true) or `off` (false).

An example of defining and using a **boolean** option follows:

```
option use-zephyr code 180 = boolean;
option use-zephyr on;
```

### 3.6.10.2 Integer

**option** <new-name> **code** <new-code> = [ <sign> ] **integer** <width>;

The <sign> token should either be blank, unsigned or signed. The width can be either 8, 16 or 32, and refers to the number of bits in the integer.

An example of defining and using an **integer** option follows:

```
option sql-connection-max code 192 = unsigned integer 16;
option sql-connection-max 1536;
```

### 3.6.10.3 IPv4 address

**option** <new-name> **code** <new-code> = **ip-address**;

An option whose structure is an IP address can be expressed either as a domain name or as a dotted quad.

An example of defining and using an **ip-address** option follows:

```
option sql-server-address code 193 = ip-address;
option sql-server-address sql.example.com;
```

### 3.6.10.4 IPv6 address

**option** <new-name> **code** <new-code> = **ip6-address**;

An option whose structure is an IPv6 address must be expressed as a valid IPv6 address.

An example of defining and using an **ip6-address** option follows:

```
option dhcp6.some-server code 1234 = array of ip6-address;  
option dhcp6.some-server 3ffe:bbbb:aaaa:aaaa::1, ↵  
↪3ffe:bbbb:aaaa:aaaa::2;
```

---

**Note:** In this example, the **array** option type is also used. See the **array** option type definition in this manual page.

---

### 3.6.10.5 Text

**option** <new-name> **code** <new-code> = **text**;

An option whose type is text will encode an ASCII text string.

An example of defining and using a **text** option follows:

```
option sql-default-connection-name code 194 = text;  
option sql-default-connection-name "PRODZA";
```

### 3.6.10.6 Data string

**option** <new-name> **code** <new-code> = **string**;

An option whose type is a data string is essentially just a collection of bytes, and can be specified either as quoted text, like the text, or as a list of hexadecimal contents separated by colons whose must be between 0 and FF.

An example of defining and using a **string** option follows:

```
option sql-identification-token code 195 = string;  
option sql-identification-token 17:23:19:a6:42:ea:99:7c:22;
```

### 3.6.10.7 Domain list

**option** <new-name> **code** <new-code> = **domain-list** [ **compressed** ];

An option whose type is **domain-list** is a list of domain names in [RFC 1035](#) wire format, separated by root labels. The optional **compressed** keyword indicates if the option should be compressed relative to the start of the option contents (not the packet contents).

---

**Note:** When in doubt, omit the **compressed** keyword. When the software receives an option that is compressed and the **compressed** keyword is omitted, it will still decompress the option (relative to the option contents field). The keyword only controls whether or not transmitted packets are compressed.

---

---

**Note:** When domain-list formatted options are output as environment variables to *dhclient-script*(8), the standard DNS escape mechanism is used: they are decimal. For example, this is appropriate for direct use in */etc/resolv.conf*.

---

### 3.6.10.8 Encapsulation

**option** <new-name> **code** <new-code> = **encapsulate** <identifier>;

An option whose type is **encapsulate** will encapsulate the contents of the option space specified in <identifier>. Examples of encapsulated options in the DHCP protocol as it currently exists include the **vendor-encapsulated-options** option, the **netware-suboptions** option, and the **relay-agent-information** option.

An example of defining and using an **encapsulate** option follows:

```
option space local;
option local.demo code 1 = text;
option local-encapsulation code 197 = encapsulate local;
option local.demo "demo";
```

### 3.6.10.9 Arrays

**option** <new-name> **code** <new-code> = **array of** <type>;

Options can contain arrays of any of the above types except for the **text** and **string** types, which aren't currently supported in arrays.

An example of defining and using **array of** follows:

```
option kerberos-servers code 200 = array of ip-address;
option kerberos-servers 10.20.10.1, 10.20.11.1;
```

### 3.6.10.10 Records

Options can also contain data structures consisting of a sequence of data types, which is sometimes called a record type.

An example of defining and using a record type follows:

```
option contrived-001 code 201 = { boolean, integer 32, text };
option contrived-001 on 1772 "contrivance";
```

It is also possible to have options that are arrays of records, for example:

```
option new-static-routes code 201 = array of {
    ip-address, ip-address, ip-address, integer 8
};

option static-routes
    10.0.0.0 255.255.255.0 net-0-rtr.example.com 1,
    10.0.1.0 255.255.255.0 net-1-rtr.example.com 1,
    10.2.0.0 255.255.224.0 net-2-0-rtr.example.com 3;
```

### 3.6.11 Vendor encapsulated options

The DHCP protocol defines the **vendor-encapsulated-options** option, which allows vendors to define their own options that will be sent encapsulated in a standard DHCP option. It also defines the Vendor Identified Vendor Sub Options option ("VIVSO"), and the DHCPv6 protocol defines the Vendor-specific Information Option ("VSIO"). The format of all of these options is usually internally a string of options, similarly to other normal DHCP options. The VIVSO and VSIO options differ in that they contain options that correspond to vendor Enterprise-ID numbers (assigned by IANA), which then contain options according to each Vendor's specifications. The vendor's documentation will have to be consulted in order to form options to their specification.

The value of these options can be set in one of two ways. The first way is to simply specify the data directly, using a text string or a colon-separated list of hexadecimal values. For help in forming these strings, please refer to [RFC 2132](#) for the DHCPv4 Vendor Specific Information Option, [RFC 3925](#) for the DHCPv4 Vendor Identified Vendor Sub Options, or [RFC 3315](#) for the DHCPv6 Vendor-specific Information Option.

For example:

```
option vendor-encapsulated-options
    2:4:
        AC:11:41:1:
    3:12:
        73:75:6e:64:68:63:70:2d:73:65:72:76:65:72:31:37:2d:31:
    4:12:
        2f:65:78:70:6f:72:74:2f:72:6f:6f:74:2f:69:38:36:70:63;
option vivso
```

(continues on next page)

(continued from previous page)

```

00:00:09:bf:0E:
    01:0c:
        48:65:6c:6c:6f:20:77:6f:72:6c:64:21;
option dhcp6.vendor-opts
    00:00:09:bf:
        00:01:00:0c:
            48:65:6c:6c:6f:20:77:6f:72:6c:64:21;

```

The second way of setting the value of these options is to have the DHCP server generate a vendor-specific option buffer. To do this, you must do four things: define an option space, define some options in that option space, provide values for them, and specify that that option space should be used to generate the relevant option.

To define a new option space in which vendor options can be stored, use the option space statement:

```
option space <name> [ [ code width <number> ] [ length width <number> ] [ hash size <number> ] ] ;
```

The numbers following **code width**, **length width**, and **hash size** respectively identify the number of bytes used to describe option codes, option lengths, and the size in buckets of the hash tables to hold options in this space. Most DHCPv4 option spaces use 1 byte codes and lengths, which is the default, whereas most DHCPv6 option spaces use 2 byte codes and lengths.

The code and length widths are used in DHCP protocol --- you must configure these numbers to match the applicable option space you are configuring. They each default to 1. Valid values for code widths are 1, 2 or 4. Valid values for length widths are 0, 1 or 2. Most DHCPv4 option spaces use 1 byte codes and lengths, which is the default, whereas most DHCPv6 option spaces use 2 byte codes and lengths. A zero-byte length produces options similar to the DHCPv6 Vendor-specific Information Option, but not their contents!

The hash size defaults depend upon the code width selected, and may be 254 or 1009. Valid values range between 1 and 65535.

**Warning:** The higher you configure the hash size, the more memory will be used. It is considered good practice to configure a value that is slightly larger than the estimated number of options you plan to configure within the space.

The name can then be used in option definitions, as described earlier in this document. For example:

```

option space SUNW code width 1 length width 1 hash size 3;
option SUNW.server-address code 2 = ip-address;
option SUNW.server-name code 3 = text;
option SUNW.root-path code 4 = text;

```

(continues on next page)



(continued from previous page)

```

option space BANU code width 1 length width 1 hash size 3;
option BANU.sample code 1 = text;
option vendor.BANU code 2495 = encapsulate vivso-sample;
option vendor-class.BANU code 2495 = text;

option BANU.sample "configuration text here";
option vendor-class.BANU "vendor class here";

option space docsis code width 2 length width 2 hash size 17;
option docsis.tftp-servers code 32 = array of ip6-address;
option docsis.cablelabs-configuration-file code 33 = text;
option docsis.cablelabs-syslog-servers code 34 = array of ip6-
→address;
option docsis.device-id code 36 = string;
option docsis.time-servers code 37 = array of ip6-address;
option docsis.time-offset code 38 = signed integer 32;
option vsio.docsis code 4491 = encapsulate docsis;

```

Once you have defined an option space and the format of some options, you can set up scopes that define values for those options, and you can say when to use them. For example, suppose you want to handle two different classes of clients. Using the option space definition shown in the previous example, you can send different option values to different clients based on the **vendor-class-identifier** option that the clients send, as follows:

```

class "vendor-classes" {
    match option vendor-class-identifier;
}

subclass "vendor-classes" "SUNW.Ultra-5_10" {
    vendor-option-space SUNW;
    option SUNW.root-path "/export/root/sparc";
}

subclass "vendor-classes" "SUNW.i86pc" {
    vendor-option-space SUNW;
    option SUNW.root-path "/export/root/i86pc";
}

option SUNW.server-address 172.17.65.1;
option SUNW.server-name "sundhcp-server17-1";

option vivso-sample.sample "Hello world!";

option docsis.tftp-servers ::1;

```

As you can see in the preceding example, regular scoping rules apply, so you can define values that are global in the global scope, and only define values that are specific to a particular class in the local scope. The vendor-option-space declara-

tion tells the DHCP server to use options in the SUNW option space to construct the DHCPv4 **vendor-encapsulated-options** option. This is a limitation of that option --- the DHCPv4 VIVSO and the DHCPv6 VSIO options can have multiple vendor definitions all at once (even transmitted to the same client), so it is not necessary to configure this.

### 3.6.12 See also

*dhcpd.conf(5)*, *dhcpd.leases(5)*, *dhclient.conf(5)*, *dhcp-eval(5)*,  
*dhcpd(8)*, *dhclient(8)*

### 3.6.13 Copyright

Copyright (C) 2025 Banu Systems Private Limited. All rights reserved.

Copyright (c) 2012-2016 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 2004-2010 by Internet Systems Consortium, Inc. ("ISC").

Copyright (c) 1996-2003 by Internet Software Consortium.

## RELEASE NOTES

This documentation corresponds to Lease version 1.99.14.20260521081756.d9d7a78607.

### 4.1 Lease 1.99.14

The following are release notes for Lease 1.99.14:

- RT1760: Use `IP_TOS_DSCP_EF` instead of `IP_TOS_LOWDELAY`
- RT1760: Set DF bit on outgoing datagrams over LPF/BPF
- RT1760: Use a flag to indicate fallback interface instead of string compares
- RT1760: Rename taskmgr worker threads to "work-%hu"
- RT1760: Further cleanup redundant `errno` to result conversion
- RT1760: Remove the redundant `uerr2isc()`
- RT1760: Remove obsolete `IGNORE_HOSTUNREACH` handling
- RT1760: Fix argument to `SO_BINDTODEVICE` which should be the interface name
- RT1922: Add CLI args for network I/O method selection
- RT1760: Send responses to `yiaddr` when client isn't requesting broadcast
- RT1760: Set `BIOCSDIRECTION` on BPF sockets
- RT1760: Fix interface return in the BPF implementation
- RT1921: Compile multiple network I/O methods
- RT1760: Fix interface selection code when binding to `INADDR_ANY`
- RT1760: Use `ISC_LIST` instead of custom list implementation
- RT1760: Use `loop_refcount` for references in Lease socket code
- RT1760: Send response to source IP if client knows its address
- RT1760: Fix BPF implementation

- RT1760: Set interface index in the interface struct
- RT1760: Update text in *dhcpcd(8)* manpage
- RT1760: Cleanup BPF implementation
- RT1760: Allow `IFT_LOOP` for testing on FreeBSD
- RT1760: Add missing includes for FreeBSD
- RT1920: Implement socket multi-interface support for Linux and FreeBSD
- RT1760: Save the interface index
- RT1760: Drop Solaris-specific interface scanning code
- RT1760: Indent comments properly
- RT1760: Remove SCO-specific `IP_BROADCAST_IF` socketopt
- RT1760: Remove `cdefs.h`
- RT1760: Remove `osdep.h`
- RT1760: Remove `site.h`
- RT1760: Stop using obsolete byte order macros
- RT1760: Remove obsolete comment
- RT1760: Cleanups
- RT1760: Remove obsolete macros
- RT1760: Use `FD_CLOEXEC` instead of the value 1 in `fcntl()` calls
- RT1760: Remove support for FDDI
- RT1760: Remove support for Token Ring
- RT1760: Remove `netinet` headers from tree
- RT1760: Add brackets around return argument
- RT1760: Add brackets around `sizeof` operand
- RT1760: Update coding style of the Lease code to the Loop style
- RT1760: Delete trailing whitespace
- RT1760: Update text on `db-time-format`
- RT1760: Remove unused macros and `protos`
- RT1760: Remove obsolete local definition of `SHUT_RD`
- RT1760: Remove `snprintf()/vsprintf()` hacks
- RT1760: Update message about `authoring-byte-order`
- RT1760: Remove LDAP support
- RT1760: Add back all of the socket IO code for use in testing

- RT1760: Add comment
- RT1760: Remove obsolete macros
- RT1760: Always build DNS zone lookup support
- RT1760: Remove remnants of `DDNS_UPDATE_STYLE_AD_HOC`
- RT1760: Always build DNS UPDATE support
- RT1760: Always include pid in log messages
- RT1760: Update `configure` macro message
- RT1760: Always build delayed ack support
- RT1760: Turn off delayed ack if the delayed-ack count is configured as 0
- RT1760: Modify code to support DHCP4o6 even when delayed acks are enabled
- RT1760: Always build failover protocol support
- RT1760: Always check if byteorder of the secs field is swapped
- RT1918: Fix section title syntax error

## 4.2 Lease 1.99.13

The following are release notes for Lease 1.99.13:

- RT1909: Remove reference count debugging code: this relic from another era just cluttered up the code everywhere; we will also switch to using atomic refcounts in a future release
- RT1760: Remove unnecessary macros
- RT1908: Remove tracing code from Lease: this relic from another era just cluttered up the code everywhere. Modern debugging tools do a better job.
- RT1907: Refactor all of the TSIG algorithm parsing code: this change makes Loop use a single consistent set of TSIG algorithms, and also adds support for HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, HMAC-SHA512 to the DHCP DDNS code
- RT1760: Fix fallthrough warnings
- RT1760: Fix warnings about format string conversion specifiers
- RT1760: Fix warning about `snprintf()` truncation
- RT1760: Fix warnings about incompatible function types
- RT1760: Fix shadowed variable warnings
- RT1760: Don't pass the format string
- RT1760: Move declarations to top

- RT1760: Fix warnings about const discards
- RT1760: Fix cast from ptr to int
- RT1760: Remove unused functions
- RT1760: Fix warning about `strncpy()` overflow
- RT1760: Fix warnings about returning aggregates
- RT1760: Fix warnings in `dhcrelay`
- RT1760: Fix signed vs. unsigned comparisons
- RT1760: Remove old-style function definitions
- RT1760: Make functions static

A *lot* of code cleanup and refactoring was done in this release. More to come.

## 4.3 Lease 1.99.12

The following are release notes for Lease 1.99.12:

- RT1875: Add missing cocci files to dist
- RT1870: Fix `configure.ac` warnings
- RT1869: Update main copyrights to 2026

## 4.4 Lease 1.99.11

The following are release notes for Lease 1.99.11:

- RT1840: Fix release notes text

## 4.5 Lease 1.99.10

The following are release notes for Lease 1.99.10:

- RT1819: Fix Debian package homepage URLs

## 4.6 Lease 1.99.9

The following are release notes for Lease 1.99.9:

- RT1812: Add Debian and Ubuntu to supported platforms
- RT1801: Fix warning about possible thread name buffer overflow (`pthread_setname_np()`)
- RT1760: Remove socket interface code
- RT1760: Remove unused socket code
- RT1760: Remove raw socket implementation code
- RT1760: Remove log messages about HP JetAdmin software
- RT1760: Update text in manpage
- RT1760: Remove NIT code
- RT1760: Remove DLPI code
- RT1760: Remove UPF code
- RT1760: Move `icmp.c` function prototypes to `icmp.h`
- RT1760: Remove local version of `inet_aton()`
- RT1776: Use separate heap indexes for tracking active and inactive heaps
- RT1777: Rename heap callback function types and args
- RT1772: Refactor and port `mdb6_unittest.c`
- RT1775: Save and restore lease's heap index
- RT1774: Remove CYGWIN specific code
- RT1771: Refactor and port `duid_unittest.c`
- RT1770: Refactor and port `leaseq_unittest.c`
- RT1769: Refactor and port `simple_unittest.c`
- RT1768: Refactor and port `load_bal_unittest.c`
- RT1767: Refactor and port `hash_unittest.c`
- RT1766: Make `libdhclient` library
- RT1765: Make `libdhcpcd` library
- RT1764: Refactor and port `test_alloc.c`
- RT1763: Refactor and port `ns_name_test.c`
- RT1762: Refactor and port `misc_unittest.c`
- RT1761: Refactor and port `dns_unittest.c`
- RT1760: Move `ddns.c` function prototypes to `ddns.h`

- RT1760: Move `dns.c` function prototypes to `dns.h`
- RT1760: Make `repudiate_zone()` statically scoped
- RT1760: Make `dns_zone_lookup()` statically scoped
- RT1760: Refactor `dispatch()`
- RT1760: Add include
- RT1760: Fix module name, etc.
- RT1760: Prefix lease to function names
- RT1760: Delete trailing whitespace
- RT1760: Fix old style function decl
- RT1760: Fix function args
- RT1760: Cleanup includes
- RT1760: Replace defines with enum
- RT1760: Delete trailing whitespace
- RT1760: Make variables static scoped
- RT1760: Fix no previous prototype compiler warnings
- RT1755: Add Debian packaging
- RT1746: Fix user creation in RPM spec file for new RPM versions
- RT1743: Disable parallel make in docs directories when using Sphinx
- RT1742: Add `installation.rst` to `EXTRA_DIST`
- RT1740: Update list of supported platforms
- RT1738: Use standard C integer types
- RT1737: Move `leasechain` prototypes to `leasechain.h`
- RT1736: Remove unused `resolv.c`

## 4.7 Lease 1.99.8

The following are release notes for Lease 1.99.8:

- RT1671: `libevent` and `libzip` deps are limited to just the border RPM package now.



## 4.8 Lease 1.99.7

The following are release notes for Lease 1.99.7:

- RT1428: The packaging of Lease was updated. Lease is part of a source code tree with other components such as Loop and Border, and previously only Lease and Loop were packaged. Other components of the tree are also packaged now as part of a multi-package RPM spec file. The `border-release` package which installed the `border-epel`, `border-epel-testing`, `border-fedora`, and `border-fedora-testing` DNF package repositories and their package signing PGP key has been replaced with the `akira-release` package which installs the `akira-epel`, `akira-epel-testing`, `akira-fedora`, and `akira-fedora-testing` DNF package repositories and their package signing PGP key.
- RT1672: A minor syntax error was fixed in the Lease RPM's post-installation script (it did not cause an actual issue other than the error report).
- RT1613: All the programs of Loop and Lease were updated to use consistent command line options for version, verbosity, help, etc.
- RT1618: DNS names were added to the list of items in the data and privacy section of the Loop and Lease user manuals.

To upgrade from a previous release of Lease on an RPM platform, first remove the obsolete `border-release` RPM:

```
$ sudo dnf remove border-release
```

Then, install the `akira-release` package as described in the installation instructions (see the chapter titled *Installation*). This will install the new DNF package repositories.

Then, upgrade the `lease` package, which should install it from the new DNF package repository:

```
$ sudo dnf upgrade lease
```

You may clean up the old Border Package Signer key using `clean-rpm-gpg-pubkey`:

```
$ sudo dnf install clean-rpm-gpg-pubkey
$ sudo clean-rpm-gpg-pubkey
```

## 4.9 Lease 1.99.6

This is the first release of Lease. Lease has been a part of a source code tree with other dependency components such as Loop for some time now, where it had been maintained with the extensive code refactoring that was done in the tree. It was not packaged for release previously.

The following are release notes for Lease 1.99.6:

- RT1606: Lease packages are now built. Lease is part of a source code tree with other components such as Loop, and previously only Loop was packaged. Other components of the tree are also packaged now as part of a multi-package RPM spec file.

To install Lease, please see the chapter titled *Installation*.

## 4.10 Lease version numbering scheme

Lease version numbers have the grammar <MAJOR>.<MINOR>.<PATCH>.<COMMIT-TIMESTAMP>.<COMMIT-HASH>. The *MAJOR* and *MINOR* version numbers together represent a source code branch of Lease (see *Lease branches*).

- The *MAJOR* version number is incremented when configuration options, API, and behavior of features change compared to the existing version. Switching to a new *MAJOR* version may require modifying existing config files to make them compatible with the new version.
- The *MINOR* version number is incremented when new configuration options, API, and features are introduced that are compatible with existing configuration options. Switching to a new *MINOR* version will not require modifying existing config files to make them compatible with the new version.
- The *PATCH* version number is incremented when only bugs have been fixed in a new version. Switching to a new *PATCH* version will not require modifying existing config files to make them compatible with the new version.
- The *COMMIT-TIMESTAMP* field is auto-generated and contains the UTC timestamp of the source code commit from which the Lease release was made. The timestamp value is formatted as YYYYMMDDHHMMSS, as the output of:

```
date +%Y%m%d%H%M%S
```

- The *COMMIT-HASH* field is auto-generated and contains the abbreviated commit hash of the source code commit from which the Lease release was made. The hash value is formatted as the output of:

```
git log -n1 --reverse --pretty=%h
```

For example,

- If you're upgrading from version 1.2.1 to version 1.4.0, Lease's config files should not require any changes. You may also check for new features that have become available in the 1.4 branch.
- If you're upgrading from version 1.2.1 to version 2.0.0, it is possible that some of the contents of your existing config files may need changes. You may also check for new features that have become available in the 2.0 branch.
- If you're upgrading from version 1.2.1 to version 1.2.4, Lease's config files should not require any changes. The newer version only contains bugfixes.

---

**Note:** During a major version's release series, features and/or programs scheduled for removal in the next major release may be marked as deprecated. They will however still be supported until the end-of-life of that major release.

---

### 4.10.1 Stable and development versions

Even-numbered *minor* versions indicate stable branch releases, whereas odd-numbered *minor* versions indicate development branch releases. For example,

- 1.2.0 is a stable branch release,
- 1.3.0 is a development branch release,
- 1.99.0 is a development branch release,
- 2.0.3 is a stable branch release, and
- 2.1.1 is a development branch release.

**Warning:** Development branch releases should not be used in production as their features and interfaces may change. Development branch releases may not work properly, may have unexpected behaviors, may crash, etc.

## 4.11 Lease branches

The following is information on current Lease branches.

Branch	Type	First release date	End-of-life date
1.99	Development	2024-12-10	To be announced
2.0	Stable	To be announced	To be announced

Development branches have no planned end-of-life. Typically, development on such branches is stopped when a new *MINOR*+1 or *MAJOR*+1 stable branch is created off it.

## 4.12 History of Lease

ISC DHCP was originally written by Ted Lemon under a contract with Vixie Labs with the goal of being a complete reference implementation of the DHCP protocol. Funding for this project was provided by Internet Systems Consortium. The first release of the ISC DHCP distribution in December 1997 included just the DHCP server. Release 2 in June 1999 added a DHCP client and a BOOTP/DHCP relay agent. DHCP 3 was released in October 2001 and included DHCP failover support, OMAPI, Dynamic DNS, conditional behaviour, client classing, and more. Version 3 of the DHCP server was funded by Nominum, Inc. The 4.0 release in December 2007 introduced DHCPv6 protocol support for the server and client.

After the project was abandoned upstream, Lease was started by a former Infoblox NIOS developer with the goal of upgrading the codebase to modern standards, and integrating better with Loop (which is a dependency of Lease). Lease forked from the last ISC-licensed ISC-DHCP codebase. While externally it resembles ISC-DHCP with its similar configuration language and programs, its code has undergone considerable changes and continues to evolve at a high rate.

**LICENSE**

Copyright (C) 2024-2026 Banu Systems Private Limited. All rights reserved.

This product is not open source software. Permission is not granted to redistribute this product.

A proprietary software license is used as an overall license for distributing binaries, documentation, and other works of this product.

THE SOFTWARE IS PROVIDED "AS IS" AND BANU DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL BANU BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Portions of this product are covered by one or more of the following original copyright and license notices. This product cannot re-license the original unmodified effects and such portions of the product are covered by their original licenses listed below.

---

Copyright (c) 2004-2018 by Internet Systems Consortium, Inc. ("ISC") Copyright (c) 1995-2003 by Internet Software Consortium

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND ISC DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL ISC BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

Copyright (C) 1996-2001 Nominum, Inc. Copyright (C) 2000, 2001, 2004-2015 Nominum, Inc.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND NOMINUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL NOMINUM BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

Copyright (C) 1995-2000 by Network Associates, Inc.

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND ISC AND NETWORK ASSOCIATES DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL ISC BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

Copyright (C) 2002 Stichting NLnet, Netherlands, [stichting@nlnet.nl](mailto:stichting@nlnet.nl).

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND STICHTING NLNET DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL STICHTING NLNET BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

Copyright (c) 1987, 1990, 1993, 1994 The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

Copyright (C) The Internet Society 2005. This version of this module is part of RFC 4178; see the RFC itself for full legal notices.

(The above copyright notice is per RFC 3978 5.6 (a), q.v.)

---

Copyright (c) 2004 Masarykova universita (Masaryk University, Brno, Czech Republic)  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,

---

BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

Copyright (c) 1997 - 2003 Kungliga Tekniska Hogskolan (Royal Institute of Technology, Stockholm, Sweden). All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the Institute nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

Copyright (c) 1998 Doug Rabson All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.



2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

Copyright ((c)) 2002, Rice University All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Rice University (RICE) nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by RICE and the contributors on an "as is" basis, without any representations or warranties of any kind, express or implied including, but not limited to, representations or warranties of non-infringement, merchantability or fitness for a particular purpose. In no event shall RICE or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

---

Copyright (c) 1993 by Digital Equipment Corporation.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies, and that the name of Digital Equipment Corporation not be used in advertising or publicity pertaining to distribution of the document or software without specific, written prior permission.

---

THE SOFTWARE IS PROVIDED "AS IS" AND DIGITAL EQUIPMENT CORP. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL DIGITAL EQUIPMENT CORPORATION BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

Copyright 2000 Aaron D. Gifford. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR(S) AND CONTRIBUTOR(S) ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR(S) OR CONTRIBUTOR(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

Copyright (c) 1998 Doug Rabson. Copyright (c) 2001 Jake Burkholder. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

Copyright (c) 1999-2000 by Nortel Networks Corporation

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND NORTEL NETWORKS DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT

---

SHALL NORTEL NETWORKS BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

Copyright (c) 2000-2002 Japan Network Information Center. All rights reserved.

By using this file, you agree to the terms and conditions set forth below.

#### LICENSE TERMS AND CONDITIONS

The following License Terms and Conditions apply, unless a different license is obtained from Japan Network Information Center ("JPNIC"), a Japanese association, Kokusai-Kougyou-Kanda Bldg 6F, 2-3-4 Uchi-Kanda, Chiyoda-ku, Tokyo 101-0047, Japan.

1. Use, Modification and Redistribution (including distribution of any modified or derived work) in source and/or binary forms is permitted under this License Terms and Conditions.
2. Redistribution of source code must retain the copyright notices as they appear in each source code file, this License Terms and Conditions.
3. Redistribution in binary form must reproduce the Copyright Notice, this License Terms and Conditions, in the documentation and/or other materials provided with the distribution. For the purposes of binary distribution the "Copyright Notice" refers to the following language: "Copyright (c) 2000-2002 Japan Network Information Center. All rights reserved."
4. The name of JPNIC may not be used to endorse or promote products derived from this Software without specific prior written approval of JPNIC.
5. Disclaimer/Limitation of Liability: THIS SOFTWARE IS PROVIDED BY JPNIC "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JPNIC BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

---

Copyright (C) 2004 Nominet, Ltd.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND NOMINET DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL ISC BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

Portions Copyright RSA Security Inc.

License to copy and use this software is granted provided that it is identified as "RSA Security Inc. PKCS #11 Cryptographic Token Interface (Cryptoki)" in all material mentioning or referencing this software.

License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Security Inc. PKCS #11 Cryptographic Token Interface (Cryptoki)" in all material mentioning or referencing the derived work.

RSA Security Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

---

Copyright (c) 1996, David Mazieres <[dm@uun.org](mailto:dm@uun.org)> Copyright (c) 2008, Damien Miller <[djm@openbsd.org](mailto:djm@openbsd.org)> Copyright (c) 2013, Markus Friedl <[markus@openbsd.org](mailto:markus@openbsd.org)> Copyright (c) 2014, Theo de Raadt <[deraadt@openbsd.org](mailto:deraadt@openbsd.org)>

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

Copyright (c) 2000-2001 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
-

3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.OpenSSL.org/>)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact [licensing@OpenSSL.org](mailto:licensing@OpenSSL.org).
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.OpenSSL.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

Copyright (c) 1995, 1997, 1998 The NetBSD Foundation, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC. AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.





## **DATA AND PRIVACY**

Lease implements---as part of its software features---support for logging various types of activity for the administrator to use.

- Logging may be performed on behalf of the administrator on the machine where the Lease software is installed.
- Log messages may contain personal data about a user such as, but not limited to, IP addresses, MAC addresses, hostnames, DNS names, TSIG key names, timestamps, etc.

We note that:

- Lease does not share user data or logs with Banu Systems Private Limited, unless the administrator explicitly configures it to do so.
- Lease does not share user data or logs with other companies, organizations, or persons, unless the administrator explicitly configures it to do so.



## Symbols

- l  
command line option, 24
- 4  
command line option, 9, 23, 36
- 4o6 <port>  
command line option, 9, 24
- 6  
command line option, 23, 36
- A <length>  
command line option, 37
- D  
command line option, 37
- D <LL|LLT>  
command line option, 27
- I  
command line option, 26, 39
- N  
command line option, 27
- P  
command line option, 26
- R  
command line option, 26
- S  
command line option, 26
- T  
command line option, 10, 26
- U <ifname>  
command line option, 38
- V  
command line option, 10, 26, 37, 40
- dad-wait-time <seconds>  
command line option, 27
- early-chroot  
command line option, 10
- no-pid  
command line option, 11, 27, 37
- a  
command line option, 37
- c <count>  
command line option, 36
- cf <config-file>  
command line option, 10, 27
- chroot <directory>  
command line option, 10
- d  
command line option, 9, 24, 36
- df <duid-lease-file>  
command line option, 27
- e <VAR>=<value>  
command line option, 24
- f  
command line option, 9
- g <relay>  
command line option, 26
- group <group>  
command line option, 10
- h  
command line option, 10, 24, 36, 40
- i  
command line option, 26
- i <ifname>  
command line option, 37
- id <ifname>  
command line option, 38
- iu <ifname>  
command line option, 37
- l [<address>%]<ifname>[#<index>]  
command line option, 39
- lf <lease-file>  
command line option, 11, 27
- m <append|replace|forward|discard>  
command line option, 38
- n

command line option, [24](#)  
-nio <method>  
command line option, [9, 25, 36](#)  
-nw  
command line option, [24](#)  
-p <port>  
command line option, [9, 25, 36](#)  
-pf <pid-file>  
command line option, [11, 27, 37](#)  
-q  
command line option, [10, 24, 37](#)  
-r  
command line option, [24](#)  
-s <address>  
command line option, [9](#)  
-s <server-address>  
command line option, [25](#)  
-s <subscriber-id>  
command line option, [39](#)  
-sf <script-file>  
command line option, [27](#)  
-t  
command line option, [10](#)  
-u [<address>%]<ifname>  
command line option, [39](#)  
-user <user>  
command line option, [10](#)  
-v  
command line option, [24](#)  
-w  
command line option, [24](#)  
-x  
command line option, [25](#)

## C

command line option

-l, [24](#)  
-4, [9, 23, 36](#)  
-4o6 <port>, [9, 24](#)  
-6, [23, 36](#)  
-A <length>, [37](#)  
-D, [37](#)  
-D <LL|LLT>, [27](#)  
-I, [26, 39](#)  
-N, [27](#)  
-P, [26](#)  
-R, [26](#)  
-S, [26](#)

-T, [10, 26](#)  
-U <ifname>, [38](#)  
-V, [10, 26, 37, 40](#)  
--dad-wait-time <seconds>, [27](#)  
--early-chroot, [10](#)  
--no-pid, [11, 27, 37](#)  
-a, [37](#)  
-c <count>, [36](#)  
-cf <config-file>, [10, 27](#)  
-chroot <directory>, [10](#)  
-d, [9, 24, 36](#)  
-df <duid-lease-file>, [27](#)  
-e <VAR>=<value>, [24](#)  
-f, [9](#)  
-g <relay>, [26](#)  
-group <group>, [10](#)  
-h, [10, 24, 36, 40](#)  
-i, [26](#)  
-i <ifname>, [37](#)  
-id <ifname>, [38](#)  
-iu <ifname>, [37](#)  
-l [<address>%]<ifname>[#<index>],  
[39](#)  
-lf <lease-file>, [11, 27](#)  
-m <append|replace|forward|discard>,  
[38](#)  
-n, [24](#)  
-nio <method>, [9, 25, 36](#)  
-nw, [24](#)  
-p <port>, [9, 25, 36](#)  
-pf <pid-file>, [11, 27, 37](#)  
-q, [10, 24, 37](#)  
-r, [24](#)  
-s <address>, [9](#)  
-s <server-address>, [25](#)  
-s <subscriber-id>, [39](#)  
-sf <script-file>, [27](#)  
-t, [10](#)  
-u [<address>%]<ifname>, [39](#)  
-user <user>, [10](#)  
-v, [24](#)  
-w, [24](#)  
-x, [25](#)

## R

RFC

RFC 1001, [137, 138](#)  
RFC 1002, [137, 138](#)

RFC 1035, <a href="#">81</a> , <a href="#">135</a> , <a href="#">136</a> , <a href="#">138</a> , <a href="#">158</a>	RFC 865, <a href="#">132</a>
RFC 1042, <a href="#">136</a>	RFC 868, <a href="#">143</a>
RFC 1048, <a href="#">79</a>	RFC 887, <a href="#">140</a>
RFC 1122, <a href="#">131</a> , <a href="#">139</a> , <a href="#">140</a>	RFC 893, <a href="#">143</a>
RFC 1179, <a href="#">137</a>	RFC 894, <a href="#">136</a>
RFC 1191, <a href="#">139</a>	RFC 8945, <a href="#">67</a>
RFC 1256, <a href="#">140</a>	RFC 950, <a href="#">142</a>
RFC 2131, <a href="#">81</a> , <a href="#">108</a> , <a href="#">124</a>	
RFC 2132, <a href="#">133</a> , <a href="#">135</a> , <a href="#">159</a>	
RFC 2136, <a href="#">64</a>	
RFC 2165, <a href="#">141</a>	
RFC 2242, <a href="#">139</a> , <a href="#">147</a>	
RFC 2373, <a href="#">95</a>	
RFC 2937, <a href="#">137</a>	
RFC 3011, <a href="#">142</a>	
RFC 3046, <a href="#">145</a>	
RFC 3256, <a href="#">145</a>	
RFC 3315, <a href="#">93</a> , <a href="#">94</a> , <a href="#">149</a> , <a href="#">159</a>	
RFC 3527, <a href="#">38</a>	
RFC 3633, <a href="#">71</a> , <a href="#">90</a>	
RFC 3925, <a href="#">159</a>	
RFC 3942, <a href="#">155</a>	
RFC 4361, <a href="#">26</a>	
RFC 4701, <a href="#">26</a> , <a href="#">64</a> , <a href="#">66</a>	
RFC 4702, <a href="#">26</a> , <a href="#">64</a> , <a href="#">66</a> , <a href="#">146</a>	
RFC 4703, <a href="#">64</a>	
RFC 4776, <a href="#">136</a> , <a href="#">152</a>	
RFC 4833, <a href="#">140</a> , <a href="#">143</a> , <a href="#">153</a>	
RFC 4941, <a href="#">71</a>	
RFC 4994, <a href="#">153</a>	
RFC 5192, <a href="#">139</a> , <a href="#">152</a>	
RFC 5223, <a href="#">144</a> , <a href="#">153</a>	
RFC 5417, <a href="#">132</a> , <a href="#">153</a>	
RFC 5460, <a href="#">153</a>	
RFC 5494, <a href="#">64</a>	
RFC 5905, <a href="#">139</a>	
RFC 5969, <a href="#">139</a>	
RFC 5970, <a href="#">153</a> , <a href="#">154</a>	
RFC 5986, <a href="#">144</a> , <a href="#">153</a>	
RFC 6011, <a href="#">153</a>	
RFC 6334, <a href="#">154</a>	
RFC 6440, <a href="#">154</a>	
RFC 6731, <a href="#">140</a> , <a href="#">154</a>	
RFC 6842, <a href="#">84</a>	
RFC 6939, <a href="#">154</a>	
RFC 6977, <a href="#">154</a>	
RFC 7083, <a href="#">154</a>	
RFC 7341, <a href="#">9</a> , <a href="#">24</a>	