Jagornet DHCP Server 5.0 User Guide

A. Gregory Rabil

Table of Contents

Introduction . . . 2
- Welcome . . . 2
- Architectural Overview . . . 2
- Key Capabilities & Standards Conformance . . . 2

Getting Started . . . 4
- System requirements . . . 4
- Installation . . . 4
- Running the Server . . . 4
- Testing and Simulation Tools . . . 11
- Logging . . . 12

Configuration . . . 14
- Hierarchy . . . 14
- Server Policies . . . 14
- Channel Resource Policies . . . 15
- Lease Database Policies . . . 16
- Binding Manager Policies . . . 17
- Dynamic DNS Update Policies . . . 17
- Hostname Policies . . . 22
- DHCP Protocol Policies . . . 24
- DHCPv4 Policies . . . 26
- DHCPv6 Policies . . . 27
- High Availability . . . 29
- REST API . . . 33
- Observability: Prometheus Metrics . . . 34
- DHCP Options . . . 35
- Generic Options . . . 59
- Filters . . . 70
- Links . . . 88
- Pools . . . 91
- Static Bindings . . . 96

A. Gregory Rabil greg.rabil@jagornet.com Copyright © 2026 Jagornet Technologies, LLC 2026-11-08

Introduction

Welcome

Welcome to the **Jagornet DHCP Server Version 5.0** User Guide.

Jagornet DHCP Server is an open source, enterprise-grade Dynamic Host Configuration Protocol server for IPv4 (DHCPv4) and IPv6 (DHCPv6) built on a high-throughput, non-blocking Java network engine.

For the latest releases, documentation updates, and community support:

- Website: Jagornet Technologies
- Source Code & Issue Tracking: Jagornet DHCP on GitHub
- Community Discussions: GitHub Discussions

Architectural Overview

Jagornet DHCP is designed with a **unified single-service architecture**. Unlike multi-process solutions that require deploying and orchestrating separate daemons for DHCPv4, DHCPv6, Dynamic DNS (DDNS), high availability, and REST control agents, Jagornet provides an all-in-one service:

- **Dual-Stack in One Process:** DHCPv4 and DHCPv6 engines operate concurrently within a single runtime instance.
- **Integrated Dynamic DNS:** Forward and reverse DNS updates are performed directly by the server engine without external relay daemons.
- **Native High Availability:** Active/standby state synchronization communicates peer-to-peer using high-performance gRPC.
- **High-Concurrency I/O:** Network packet processing is driven by an asynchronous, multithreaded Netty pipeline.

Key Capabilities & Standards Conformance

DHCPv4 Standards

- Full standards-compliant DHCPv4 server including BOOTP relay support.
- RFC 2131 - Dynamic Host Configuration Protocol
- RFC 2132 - DHCP Options and BOOTP Vendor Extensions

DHCPv6 Standards & Certification

- **Certified IPv6 Ready Phase II** DHCPv6 server (IPv6 Ready Logo Program).
- RFC 3315 - Dynamic Host Configuration Protocol for IPv6 (Stateful)

- RFC 3633 - IPv6 Prefix Options for DHCPv6 (Prefix Delegation)
- RFC 3646 - DNS Configuration options for DHCPv6
- RFC 3736 - Stateless Dynamic Host Configuration Protocol for IPv6

Dynamic DNS (DDNS)

- Multi-strategy forward and reverse DNS update engine with hostname validation and generation.
- RFC 4703 - Resolution of FQDN Conflicts among DHCP Clients
- RFC 4704 - The DHCPv4 Client FQDN Option

High Availability & Lease Storage

- Peer-to-peer active/standby high availability clustering powered by gRPC lease replication.
- Pluggable database persistence supporting **PostgreSQL**, **MySQL** / **MariaDB**, **SQLite**, and embedded **H2**.

Management, Observability & Cloud Operations

- **Prometheus Metrics:** Out-of-the-box `/metrics` endpoint for Prometheus scraping (lease counts, packet rates, latency, health).
- **OpenAPI 3.0 REST API:** Full programmatic administration via REST/JSON endpoints using Jersey.
- **Flexible Configuration Formats:** Author configuration files in **YAML**, **JSON**, or **XML** validated against our published XML Schema.
- **Container-Ready:** Automated OCI/Docker container images built with Jib with support for cloud subnet overrides and non-root execution.

Getting Started

System requirements

Jagornet DHCP Server requires any platform with a certified Java 21+ runtime environment.

The server has been successfully tested on Linux, Windows and Mac OS using Eclipse Temurin, Oracle JDK, and OpenJDK 21 for the respective platforms.

Installation

Java 21+ runtime must be available on the host where the Jagornet DHCP server will be installed. The JAVA_HOME environment variable should be set to the home directory of your Java 21+ JRE or JDK.

1. Create the installation directory on the host system and expand the compressed installation file for your target platform.
   - Linux/Unix/Mac
     - Installation folder: `/opt/jagornet`
     - Compressed install file: `jagornet-dhcp-5.0.0.tar.gz`
     - `JAGORNET_DHCP_HOME=/opt/jagornet/jagornet-dhcp-5.0.0`
   - Windows
     - Installation folder: `C:\jagornet`
     - Compressed install file: `jagornet-dhcp-5.0.0.zip`
     - `JAGORNET_DHCP_HOME=C:\jagornet\jagornet-dhcp-5.0.0`
2. After extracting the compressed files, a jagornet-dhcp-5.0.0 directory will be created in the installation folder, with the following contents.
   - `JAGORNET_DHCP_HOME/bin` - executable command line files. See the section on Running the Server.
   - `JAGORNET_DHCP_HOME/config` - configuration files. See the section on Configuration.
   - `JAGORNET_DHCP_HOME/db` - database directory. See the section on Binding Database.
   - `JAGORNET_DHCP_HOME/docs` - documentation files.
   - `JAGORNET_DHCP_HOME/lib` - Java library files.
   - `JAGORNET_DHCP_HOME/log` - server output files. See the section on Logging.

Running the Server

NOTE	To start the Jagornet DHCP Server, you must have superuser or administrator access rights on the host system because the server must bind to privileged port numbers 67 for DHCPv4 and 547 for DHCPv6 to service client requests. Optionally, the server

can be instructed to bind to different port numbers for testing purposes only. See the Startup Options for more information.

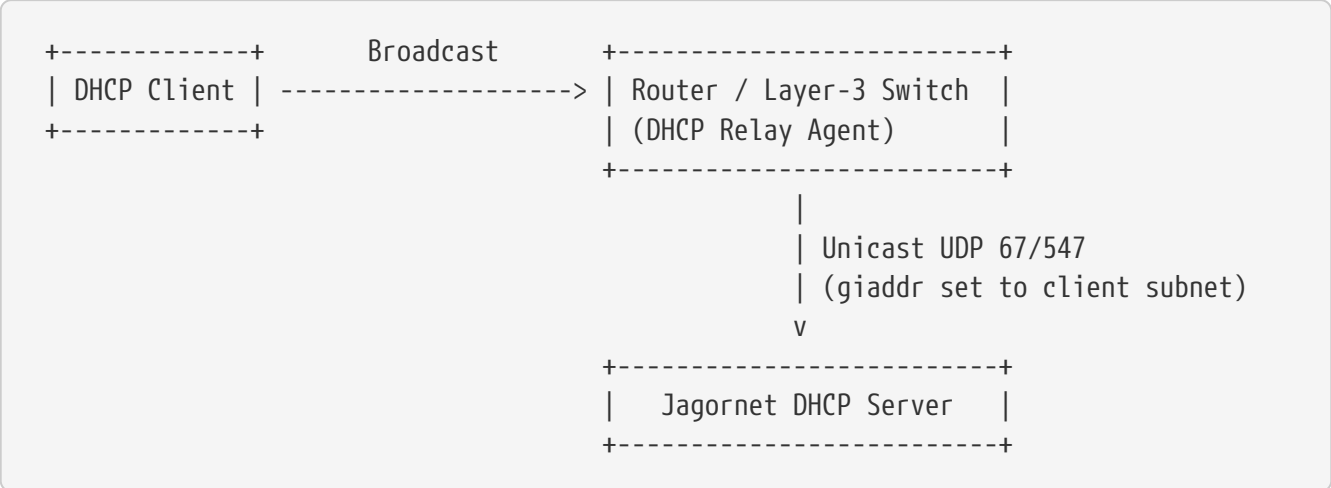
IMPORTANT	Ensure that the host system's firewall, if any, is configured to enable inbound and outbound traffic for the UDP ports of the DHCPv4 and DHCPv6 server.
IMPORTANT	Ensure that DHCP relay agents on network routers are configured to relay DHCP traffic from the client subnet(s) to the IP address of the Jagornet DHCP server. Enable DHCPv4 and DHCPv6 UDP port traffic on all intervening network equipment.

Network Deployment Models

In enterprise deployments, Jagornet DHCP Server generally operates in one of two network topologies:

1. Relayed Subnets (Standard Enterprise Model)

In most networks, the DHCP server is centralized in a management VLAN or data center, while clients reside on separate access VLANs across routers or layer-3 switches.



- **Router Configuration:** Configure your switch or router interface with an `ip helper-address` (Cisco) or `dhcp-relay` (Juniper/Arista) pointing to the Jagornet server IP.
- **Server Binding:** Start the server with `-4u <server-ip>` or bind to all unicast interfaces (default).
- **Subnet Matching:** The relay agent sets `giaddr` to the client's gateway IP. Jagornet maps `giaddr` directly against the `<address>` defined in your `link` configuration (e.g. `10.100.1.0/24`).

2. Local Broadcast Subnets (Direct Attachment)

When the server is directly attached to the same broadcast domain as the clients:

- **DHCPv4:** The server must listen for broadcast packets by specifying the interface at startup:

```
sudo ./bin/dhcpserver -4b eth0 -c config/dhcpserver.yaml
```

- **Subnet Matching:** Because `giaddr` is `0.0.0.0`, Jagornet matches the request against the link whose network CIDR matches the primary IP address configured on `eth0`.

Startup Options

The main Java class of the Jagornet DHCP Server supports the following usage:

```
usage: com.jagornet.dhcp.server.JagornetDhcpServer -c <filename> [options]
Jagornet DHCP Server 5.0.0
Copyright Jagornet Technologies, LLC
  -4b,--v4bcast <interface>         DHCPv4 broadcast interface (default = none).
                                    Use this option to specify the interface for
                                    the server to receive and send broadcast
                                    DHCPv4 packets. The default IPv4 address on
                                    the specified interface will be used for
                                    determining the DHCPv4 client link within
                                    the server configuration file.
  -4p,--v4port <portnum>            DHCPv4 Port number (default = 67).
  -4u,--v4ucast <addresses>         DHCPv4 Unicast addresses (default = all IPv4
                                    addresses). Use this option to instruct the
                                    server to bind to a specific list of IPv4
                                    addresses, separated by spaces. These
                                    addresses should be configured on one or
                                    more DHCPv4 relay agents connected to DHCPv4
                                    client links.
  -6m,--v6mcast <interfaces>        DHCPv6 Multicast interfaces (default =
                                    none). Use this option without arguments to
                                    instruct the server to bind to all
                                    multicast-enabled IPv6 interfaces on the
                                    host. Optionally, use arguments to list
                                    specific interfaces, separated by spaces.
  -6p,--v6port <portnum>            DHCPv6 Port number (default = 547).
  -6u,--v6ucast <addresses>         DHCPv6 Unicast addresses (default = all IPv6
                                    addresses). Use this option to instruct the
                                    server to bind to a specific list of global
                                    IPv6 addresses, separated by spaces. These
                                    addresses should be configured on one or
                                    more DHCPv6 relay agents connected to DHCPv6
                                    client links.
  -?,--help                         Show this help page.
  -c,--configfile <filename>        Configuration file (required: XML, JSON, or
                                    YAML).
  -ga,--gaddr <address>             gRPC address (default = all IP addresses).
                                    Use this option to instruct the server to
                                    bind to a specific IP address for gRPC
                                    communications. Set the value to 'none'
                                    (without quotes) to disable gRPC.
  -gp,--gport <portnum>             gRPC Port number (default = 9066).
  -ha,--haddr <address>             HTTPS address (default = all IP addresses).
                                    Use this option to instruct the server to
```

```
                                bind to a specific IP address for HTTPS
                                communications. Set the value to 'none'
                                (without quotes) to disable HTTPS.
  -hp,--hport <portnum>         HTTPS Port number (default = 9067).
  -tc,--test-configfile <filename>  Test configuration file, then exit.
  -v,--version                  Show version information, then exit.
```

NOTE	Users should NOT directly invoke the main Java class, but are encouraged to use the wrapper scripts which ensure the proper environment, classpath and Java VM arguments. See the following sections for Linux/Unix/Mac or Windows host systems.

Linux/Mac

The `$JAGORNET_DHCP_HOME/bin/dhcpserver` script runs the server in the foreground from a command shell (press `Ctrl+C` to stop, or see Linux Systemd Service for background daemon management). A configuration file (`-c <filename>`) is required when starting the server; invoking `dhcpserver` with no arguments displays the help summary and exits immediately.

Examples:

1. Display the server version and exit:

```
$JAGORNET_DHCP_HOME/bin/dhcpserver -v
```

2. Display the startup options and exit (instant response):

```
$JAGORNET_DHCP_HOME/bin/dhcpserver --help
```

3. Test a configuration file offline and exit:

```
$JAGORNET_DHCP_HOME/bin/dhcpserver -tc config/dhcpserver.xml
```

4. Start the server with the default configuration file:

```
$JAGORNET_DHCP_HOME/bin/dhcpserver -c config/dhcpserver.xml
```

5. Start the server on a test DHCPv4 port with broadcast on 'eth0':

```
$JAGORNET_DHCP_HOME/bin/dhcpserver -c config/dhcpserver.xml -4p 10067 -4b eth0
```

6. Start the server on a test DHCPv6 port with multicast on all IPv6 multicast interfaces:

```
$JAGORNET_DHCP_HOME/bin/dhcpserver -c config/dhcpserver.xml -6p 10547 -6m
```

7. Start the server with support for DHCPv4 broadcast on 'eth0' and DHCPv6 multicast on 'eth1':

```
$JAGORNET_DHCP_HOME/bin/dhcpserver -c config/dhcpserver.xml -4b eth0 -6m eth1
```

8. Start the server with an alternate configuration file, one unicast address, and two multicast interfaces:

```
$JAGORNET_DHCP_HOME/bin/dhcpserver -c config/my-dhcpserver.yaml -6u 2001:db8::1 -6m
eth0 eth1
```

Linux Systemd Service

Jagornet DHCP Server includes automated service installation and uninstallation scripts in `$JAGORNET_DHCP_HOME/bin/`.

The installer script handles creating the `/opt/jagornet/dhcp` symlink, creating the unprivileged `jagornet` system user and group, setting proper file permissions, creating an environment file at `/etc/default/jagornet-dhcp`, and registering and starting the systemd service.

1. Install and start the systemd service:

```
sudo $JAGORNET_DHCP_HOME/bin/install-service.sh
```

NOTE You can pass `--no-start` to install and enable the service on boot without starting it immediately: `sudo ./bin/install-service.sh --no-start`. Run `sudo ./bin/install-service.sh --help` for all options.

1. Customize the configuration file, interfaces/ports, or JVM options in `/etc/default/jagornet-dhcp`:

```
# Override configuration file path (default: /opt/jagornet/dhcp/config/dhcpserver.xml)
JAGORNET_DHCP_CONFIG=/opt/jagornet/dhcp/config/my-dhcpserver.yaml

# Specify startup options such as broadcast/multicast interfaces or ports
JAGORNET_DHCP_OPTS="-4b eth0 -6m eth0"

# Override JVM heap settings
JAVA_OPTS="-Xms512m -Xmx2g"
```

After modifying `/etc/default/jagornet-dhcp`, restart the service (`sudo systemctl restart jagornet-dhcp`) for changes to take effect.

1. Manage the service:

```
sudo systemctl status jagornet-dhcp
sudo systemctl stop jagornet-dhcp
sudo systemctl restart jagornet-dhcp
sudo journalctl -u jagornet-dhcp -f
```

2. Uninstall the service:

```
sudo $JAGORNET_DHCP_HOME/bin/uninstall-service.sh
```

NOTE	Pass `--remove-symlink` to also clean up the `/opt/jagornet/dhcp` symlink: `sudo ./bin/uninstall-service.sh --remove-symlink`. Application configuration, databases, and logs are preserved.

Windows

Command Shell

The `%JAGORNET_DHCP_HOME%\bin\dhcpserver.bat` batch file runs the server in the foreground from a command prompt (press `Ctrl+C` to stop). A configuration file (`-c <filename>`) is required when starting the server; invoking `dhcpserver.bat` with no arguments displays the help summary and exits.

Examples:

1. Display the startup options and exit:

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver.bat --help
```

2. Test a configuration file offline and exit:

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver.bat -tc config\dhcpserver.xml
```

3. Start the server with the default configuration file:

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver.bat -c config\dhcpserver.xml
```

4. Start the server with an alternate configuration file:

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver.bat -c config\my-dhcpserver.yaml
```

5. Start the server on a test DHCPv4 port with support for broadcast on the IPv4 broadcast-enabled interface named 'eth0':

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver.bat -c config\dhcpserver.xml -4p 10067 -4b eth0
```

6. Start the server on a test DHCPv6 port with support for multicast on all IPv6 multicast-enabled interfaces:

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver.bat -c config\dhcpserver.xml -6p 10547 -6m
```

7. Start the server with support for DHCPv4 broadcast on the interface named 'eth0' and DHCPv6 multicast on the interface named 'eth1':

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver.bat -c config\dhcpserver.xml -4b eth0 -6m eth1
```

8. Start the server with an alternate configuration file, one specific unicast address, and two specific multicast interfaces:

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver.bat -c config\my-dhcpserver.yaml -6u
2001:db8::1 -6m eth0 eth1
```

Windows Service (WinSW)

Jagornet DHCP Server can operate as a Microsoft Windows Service using WinSW (Windows Service Wrapper). The wrapper is pre-bundled as `%JAGORNET_DHCP_HOME%\bin\jagornet-service.exe` with its configuration `%JAGORNET_DHCP_HOME%\bin\jagornet-service.xml`.

To customize the configuration file path, interface/port startup options, or JVM heap settings for the Windows service, edit `<arguments>` in `%JAGORNET_DHCP_HOME%\bin\jagornet-service.xml` before starting or restarting the service.

1. Run `%JAGORNET_DHCP_HOME%\bin\winsw-install.bat` as Administrator to install and start the service.
2. Manage the service using Windows Services (`services.msc`) or `%JAGORNET_DHCP_HOME%\bin\jagornet-service.exe [start|stop|status]`.
3. Run `%JAGORNET_DHCP_HOME%\bin\winsw-uninstall.bat` as Administrator to remove the service.

Docker / Container Deployment

Jagornet DHCP Server is available as an optimized, multi-architecture OCI/Docker container image.

To run the container attached to the host network (required for standard broadcast and multicast DHCP traffic):

```
docker run -d \
  --name jagornet-dhcp \
  --net=host \
```

```
  -v /path/to/my/config:/app/config \
  jagornet/dhcp-server:{fullversion}
```

The container runs with `/app` as its working directory and mounts configuration files under `/app/config`. By default, the container starts with `/app/config/dhcpserver-jib.json`, which binds the server without serving any client links until custom configuration is mounted.

Testing and Simulation Tools

Jagornet DHCP Server includes built-in command-line utilities in `$JAGORNET_DHCP_HOME/bin/` to simplify host interface discovery, local configuration generation, and traffic simulation without requiring client hardware or administrator privileges.

Network Interface Discovery (`listinterfaces`)

The `listinterfaces` utility (`listinterfaces.bat` on Windows) displays the host network interfaces, hardware (MAC) addresses, and IPv4/IPv6 CIDR addresses as identified by the Java runtime. This is especially useful on Windows to see the exact interface name (e.g. `eth4`, `wlan0`) corresponding to a Windows adapter description for use with `-4b`, `-6m`, `gentestconfig -i`, or `<interface>` link configuration.

```
# List active (UP, non-loopback) network interfaces:
./bin/listinterfaces

# Include inactive (DOWN) and loopback interfaces:
./bin/listinterfaces -a
```

Test Configuration Generator (`gentestconfig`)

The `gentestconfig` utility automatically inspects active host network interfaces, discovers the local IP subnet, and generates a working configuration file tailored to the local environment.

```
# Generate a test configuration for a specific network interface:
./bin/gentestconfig -v 4 -i eth0 -f config/dhcpserver-test.xml

# List available network interfaces:
./bin/gentestconfig -l

# Show all options:
./bin/gentestconfig --help
```

Options:

- `-i, --interface <name>`: Network interface to inspect (e.g. `eth0`, `en0`). Defaults to the first active interface.
- `-v, --version <4|6|both>`: Protocol version to generate (default: `both`).

- `-f, --filename <file>`: Output configuration file path (default: `dhcpserver-test-config.xml`).
- `-l, --list-interfaces`: List active host network interfaces and exit.

DHCPv4 Client Simulator (`testclientv4`)

The `testclientv4` utility simulates DHCPv4 client requests (DISCOVER → OFFER → REQUEST → ACK → RELEASE) directly to a target server. It behaves like a DHCP relay agent communicating unicast with the server, making it possible to test lease allocation on arbitrary test ports.

```
# Run 25 client requests against an unprivileged test port:
./bin/testclientv4 -sp 10067 -n 25

# Simulate requests with FQDN / hostname options:
./bin/testclientv4 -sp 10067 -hn

# Show all options:
./bin/testclientv4 --help
```

Key Options:

- `-sp, --serverport <port>`: Destination server port (default: 67).
- `-cp, --clientport <port>`: Source port for receiving server replies. When `-sp` is set to an unprivileged port, `-cp` automatically defaults to `serverport + 1` to prevent port collisions on the same host.
- `-n, --number <count>`: Number of simulated client requests to send (default: 100).
- `-hn, --hostname`: Include DHCPv4 Host Name option in requests.
- `-x, --release`: Send DHCPRELEASE messages after acquiring leases.
- `-rr, --requestrate <rate>`: Throttle transmission to a maximum request rate per second.

DHCPv6 Client Simulator (`testclientv6`)

The `testclientv6` utility simulates DHCPv6 client requests (SOLICIT → ADVERTISE → REQUEST → REPLY) for stateful IPv6 address assignments.

```
# Run client requests against a DHCPv6 test port:
./bin/testclientv6 -p 10547 -n 10

# Show all options:
./bin/testclientv6 --help
```

Logging

Logging for the Jagornet DHCP server is performed using the Apache Log4j 2 logging logging service. The Log4j2 configuration file is located in the `config/log4j2.xml` file. The server writes all

initial startup messages to the standard output and error streams. After the server initialization, all messages will be written to the `log/dhcpserver.log` file. When the file reaches 10MB, it is "rolled" over to a new `log/dhcpserver-#.log` file up to ten files of logging history. These settings are controlled by the Log4j2 RollingFileAppender.

Configuration

Hierarchy

The Jagornet DHCP server supports configuration files in **XML** (`.xml`), **YAML** (`.yaml` or `.yml`), and **JSON** (`.json` or `.jsn`) formats. By default, the server loads `config/dhcpserver.xml`; use the `-c` / `--configfile` option (see Startup Options) to specify a YAML, JSON, or alternate XML configuration file (see `config/dhcpserver-basic. and config/dhcpserver-sample.` for examples in all three formats).

All three formats share the same structure and element names formally defined by the XML schema at `config/dhcpserver.xsd`. The configuration file has this general hierarchical structure:

- Global policies and options
- Global filters
  - Filter policies and options
- Links
  - Link policies and options
  - V4/V6address/v6prefix pools
    - Pool policies and options
    - Pool filters
      - Pool filter policies and options
  - Link filters
    - Link filter policies and options
    - Link filter v6address/v6prefix/v4 pools
      - Link filter pool policies and options

Policies and options are defined below and follow the natural hierarchy rules. That is, policies and options defined at a higher level apply to all lower levels unless override by another level in the hierarchy, which then takes precedence to the further lower levels. Options and polices cannot be removed or set to null at any level, unless specifically stated otherwise.

Server Policies

Server polices are configured using the `polices` element. Each `policyList` element consists of `name` and `value` elements. For example:

xml

```
<policies>
  <policyList>
    <name>ha.role</name>
    <value>primary</value>
  </policyList>
```

```
  <policyList>
    <name>ha.peerServer</name>
    <value>10.0.0.20</value>
  </policyList>
</policies>
```

json

```
"policies" : {
  "policyList" : [ {
    "name" : "ha.role",
    "value" : "primary"
  },
  {
    "name" : "ha.peerServer",
    "value" : "10.0.0.20"
  } ]
}
```

yaml

```
policies:
  policyList:
  - name: "ha.role"
    value: "primary"
  - name: "ha.peerServer"
    value: "10.0.0.20"
```

The sections below describe the configuration policies available for the Jagornet DHCP Server. The Hierarchy list indicates which levels of the configuration hierarchy the policy is supported. For policies that are supported at 'all' levels, the lower level policy overrides the value of any matching higher level policy.

Channel Resource Policies

Channels are used for processing all incoming network requests. The following *expert* policies can be adjusted if necessary.

Policy: channel.threadPoolSize (Thread Pool Size)

- Name: `channel.threadPoolSize`
- Default: `16`
- Description: The size of the Netty thread pool allocated for network channel request processing.
- Hierarchy: global

Policy: channel.readBufferSize (Read Buffer Size)

- Name: `channel.readBufferSize`
- Default: `307200`
- Description: The size, in bytes, of the network channel socket read buffer for receiving incoming DHCP packets.
- Hierarchy: global

Policy: channel.writeBufferSize (Write Buffer Size)

- Name: `channel.writeBufferSize`
- Default: `307200`
- Description: The size, in bytes, of the network channel socket write buffer for transmitting outgoing DHCP responses.
- Hierarchy: global

Lease Database Policies

The lease information is stored in a supported JDBC database. The following *expert* policies can be adjusted if necessary.

Policy: database.schemaType (Schema Type)

- Name: `database.schemaType`
- Default: `jdbc-h2`
- Description: The lease database schema and persistence backend type. Supported options:
  - `jdbc-h2` - uses JDBC to access an embedded H2 database for lease bindings (default).
  - `jdbc-derby` - uses JDBC to access an embedded Apache Derby database for lease bindings.
  - `jdbc-sqlite` - uses JDBC to access an embedded SQLite database for lease bindings.
  - `jdbc-postgres` - uses JDBC to access an external PostgreSQL database for lease bindings.
  - `jdbc-mysql` - uses JDBC to access an external MySQL or MariaDB database for lease bindings.
  - `jdbc-mariadb` - alias for `jdbc-mysql`.
- Hierarchy: global

Policy: database.schemaVersion (Schema Version)

- Name: `database.schemaVersion`
- Default: `2`
- Description: The lease database schema version. Version 1 schema uses the deprecated relational model and can only be used with the `jdbc-*` schema types. Version 2 uses a high-performance single-table model and can be used with all schema types.

- Hierarchy: global

Binding Manager Policies

The binding manager is responsible for lease binding lifecycle maintenance. The following *expert* policies can be adjusted if necessary.

Policy: binding.manager.reaper.startupDelay (Reaper Startup Delay)

- Name: `binding.manager.reaper.startupDelay`
- Default: `10000`
- Description: Number of milliseconds for the background reaper thread to wait after server startup before performing the initial scan for expired lease bindings. Note that bindings are always expired on demand when needed to free them for assignment.
- Hierarchy: global

Policy: binding.manager.reaper.runPeriod (Reaper Run Period)

- Name: `binding.manager.reaper.runPeriod`
- Default: `60000`
- Description: Number of milliseconds for the background reaper thread to wait between periodic checks for expired lease bindings.
- Hierarchy: global

Policy: binding.manager.offerExpiration (Offer Expiration)

- Name: `binding.manager.offerExpiration`
- Default: `12000`
- Description: Number of milliseconds after which an offered IP address is considered expired and returned to the free pool if the client does not complete the lease request.
- Hierarchy: global

Policy: binding.manager.deleteOldBindings (Delete Old Bindings)

- Name: `binding.manager.deleteOldBindings`
- Default: `false`
- Description: Flag indicating whether the server should permanently delete lease bindings upon expiration, or keep the binding record in the database marked as expired.
- Hierarchy: global

Dynamic DNS Update Policies

Jagornet DHCP Server supports standard Dynamic DNS Update mechanisms defined by the

following IETF RFCs:

- RFC 4702
- RFC 4703
- RFC 4704

The following policies configure Dynamic DNS update processing.

Policy: ddns.mode (DDNS Mode)

- Name: `ddns.mode`
- Default: `disabled`
- Description: Controls Dynamic DNS (DDNS) update delegation between the DHCP server and clients. Available values:
  - `disabled` - no DDNS updates performed.
  - `server` - server performs both forward (A/AAAA) and reverse (PTR) updates, overriding client requests if needed.
  - `client` - server honors client FQDN flags. If client requests not to update (N flag), no updates are performed. If client requests to perform forward updates (S=0), server updates only reverse (PTR) records.
- Hierarchy: all

NOTE	The legacy policy `ddns.update` (`none`, `server`, `honorNoUpdate`, `honorNoA`, `honorNoAAAA`) is retained for backward compatibility and is automatically mapped to `ddns.mode`.

Policy: ddns.record.type (DDNS Record Type)

- Name: `ddns.record.type`
- Default: `dhcid`
- Description: Specifies the identification resource record type published in forward DNS zones during DDNS updates:
  - `dhcid` - RFC 4701 standard DHCID resource record (RR type 49).
  - `txt` - Legacy ISC DHCP "interim" style TXT resource record (RR type 16), facilitating migration from ISC DHCP environments.
- Hierarchy: all

Policy: ddns.conflict.strategy (DDNS Conflict Strategy)

- Name: `ddns.conflict.strategy`
- Default: `strict`
- Description: Controls the conflict resolution strategy used during forward DNS updates:
  - `strict` - RFC 4703 client ownership. The first registrant owns the FQDN; other dynamic

clients cannot overwrite or claim it.

  - `replace-dynamic` - Dynamic client takeover with static host protection. Dynamic clients can displace other dynamic client entries (identified via a shared dynamic token), while strictly protecting static infrastructure records (servers, routers, printers) that lack the dynamic marker.
  - `unconditional` - Blind replace without prerequisites (equivalent to ISC DHCP `update-conflict-detection false`).

- Hierarchy: all

Policy: ddns.dynamic.token (DDNS Dynamic Token)

- Name: `ddns.dynamic.token`
- Default: `dynamic`
- Description: The marker string embedded in TXT or DHCID records when `ddns.conflict.strategy` is set to `replace-dynamic` to distinguish dynamic client registrations from static infrastructure.
- Hierarchy: all

Policy: ddns.synchronize (DDNS Synchronize)

- Name: `ddns.synchronize`
- Default: `false`
- Description: Flag indicating whether the DHCP server should synchronize DDNS updates with issuing of leases. When `true`, the DHCP Reply message is held until the DDNS update completes. When `false` (default), DDNS updates are queued and performed asynchronously.
- Hierarchy: all

Policy: ddns.domain (DDNS Domain)

- Name: `ddns.domain`
- Description: The default DNS domain suffix to append to unqualified client hostnames to form a Fully Qualified Domain Name (FQDN) for DDNS updates. If the client FQDN already contains a domain name, that domain name (everything after the first label) is replaced by this configured domain.
- Hierarchy: all

Policy: ddns.ttl (DDNS TTL)

- Name: `ddns.ttl`
- Default: `0.3`
- Description: Time-to-Live (TTL) for DDNS updates. If the value is less than 1.0, it is calculated as a fraction of the client lease valid lifetime in seconds (e.g. `0.3` = 30% of lease time). If 1.0 or greater, it represents an absolute number of seconds.

- Hierarchy: all

Policy: ddns.server (DDNS Server)

- Name: `ddns.server`
- Description: The IP address or hostname of the dynamic DNS server for sending DDNS updates.
- Hierarchy: all

Policy: ddns.tsig.keyName (TSIG Key Name)

- Name: `ddns.tsig.keyName`
- Description: The key name identifier for TSIG (Transaction Signature, RFC 2845) authenticated DDNS updates.
- Hierarchy: all

Policy: ddns.tsig.algorithm (TSIG Algorithm)

- Name: `ddns.tsig.algorithm`
- Description: The cryptographic algorithm name used for TSIG-signed DDNS updates. Currently supported value is `hmac-sha256`.
- Hierarchy: all

Policy: ddns.tsig.keyData (TSIG Key Data)

- Name: `ddns.tsig.keyData`
- Description: The base64-encoded shared secret key data used for TSIG-signed DDNS updates.
- Hierarchy: all

Policy: ddns.forward.zone.name (Forward Zone Name)

- Name: `ddns.forward.zone.name`
- Description: The name of the dynamic DNS zone for forward (A/AAAA) updates. If not set, the forward zone name is automatically determined in order of precedence: (1) `ddns.domain` policy value, (2) domain configured for client (Option 15 `v4DomainNameOption` or Option 24 `v6DomainSearchListOption`), or (3) domain portion of client-supplied FQDN.
- Hierarchy: all

Policy: ddns.forward.zone.ttl (Forward Zone TTL)

- Name: `ddns.forward.zone.ttl`
- Default: `0.3`
- Description: Time-to-Live (TTL) specific to forward zone DDNS updates. Overrides `ddns.ttl` for forward updates. Values < 1.0 represent lease percentage; values >= 1.0 represent absolute seconds.

- Hierarchy: all

Policy: ddns.forward.zone.server (Forward Zone Server)

- Name: `ddns.forward.zone.server`
- Description: The IP address or hostname of the dynamic DNS server for sending forward zone DDNS updates. Overrides `ddns.server` for forward updates.
- Hierarchy: all

Policy: ddns.forward.zone.tsig.keyName (Forward Zone TSIG Key Name)

- Name: `ddns.forward.zone.tsig.keyName`
- Description: The TSIG key name for signed forward zone DDNS updates. Overrides `ddns.tsig.keyName` for forward updates.
- Hierarchy: all

Policy: ddns.forward.zone.tsig.algorithm (Forward Zone TSIG Algorithm)

- Name: `ddns.forward.zone.tsig.algorithm`
- Description: The TSIG algorithm name for signed forward zone DDNS updates. Overrides `ddns.tsig.algorithm` for forward updates. Currently supported value is `hmac-sha256`.
- Hierarchy: all

Policy: ddns.forward.zone.tsig.keyData (Forward Zone TSIG Key Data)

- Name: `ddns.forward.zone.tsig.keyData`
- Description: The base64-encoded TSIG key data for signed forward zone DDNS updates. Overrides `ddns.tsig.keyData` for forward updates.
- Hierarchy: all

Policy: ddns.reverse.zone.name (Reverse Zone Name)

- Name: `ddns.reverse.zone.name`
- Description: The name of the dynamic DNS zone for reverse (PTR) updates. If not set, the reverse zone is automatically calculated from the client's subnet address and prefix length (`in-addr.arpa.` for IPv4 octet-aligned subnets, or `ip6.arpa.` for IPv6 nibble-aligned subnets).
- Hierarchy: all

Policy: ddns.reverse.zone.bitLength (Reverse Zone Bit Length)

- Name: `ddns.reverse.zone.bitLength`
- Default: `0`
- Description: The number of bits representing the subnet prefix length for calculating reverse zone names. When set to `0` (default), the subnet prefix length of the pool or link is used.

- Hierarchy: all

Policy: ddns.reverse.zone.ttl (Reverse Zone TTL)

- Name: `ddns.reverse.zone.ttl`
- Default: `0.3`
- Description: Time-to-Live (TTL) specific to reverse zone DDNS updates. Overrides `ddns.ttl` for reverse updates. Values < 1.0 represent lease percentage; values >= 1.0 represent absolute seconds.
- Hierarchy: all

Policy: ddns.reverse.zone.server (Reverse Zone Server)

- Name: `ddns.reverse.zone.server`
- Description: The IP address or hostname of the dynamic DNS server for sending reverse zone DDNS updates. Overrides `ddns.server` for reverse updates.
- Hierarchy: all

Policy: ddns.reverse.zone.tsig.keyName (Reverse Zone TSIG Key Name)

- Name: `ddns.reverse.zone.tsig.keyName`
- Description: The TSIG key name for signed reverse zone DDNS updates. Overrides `ddns.tsig.keyName` for reverse updates.
- Hierarchy: all

Policy: ddns.reverse.zone.tsig.algorithm (Reverse Zone TSIG Algorithm)

- Name: `ddns.reverse.zone.tsig.algorithm`
- Description: The TSIG algorithm name for signed reverse zone DDNS updates. Overrides `ddns.tsig.algorithm` for reverse updates. Currently supported value is `hmac-sha256`.
- Hierarchy: all

Policy: ddns.reverse.zone.tsig.keyData (Reverse Zone TSIG Key Data)

- Name: `ddns.reverse.zone.tsig.keyData`
- Description: The base64-encoded TSIG key data for signed reverse zone DDNS updates. Overrides `ddns.tsig.keyData` for reverse updates.
- Hierarchy: all

Hostname Policies

Policies controlling client hostname validation, scrubbing for DNS compliance (RFC 952, RFC 1123, RFC 1035), and automatic hostname generation. These policies can be configured globally, per link, per pool, or per filter.

Policy: hostname.mode (Hostname Mode)

- Name: `hostname.mode`
- Default: `as-is`
- Description: Hostname processing strategy for client identification and DDNS updates. Supported values:
  - `as-is` - Use client-supplied hostname/FQDN unmodified (existing default behavior).
  - `scrub` - Validate and sanitize client-supplied hostname for DNS compliance. Drops hostname if invalid.
  - `scrub-or-generate` - Validate and sanitize client hostname; if absent or invalid, generate a compliant hostname.
  - `always-generate` - Ignore client-supplied hostname and always generate a compliant hostname from template.
  - `disabled` - Disable hostname and DDNS processing for matching clients.
- Hierarchy: all

Policy: hostname.generate.template (Hostname Generation Template)

- Name: `hostname.generate.template`
- Default: `host-${ip.dashed}`
- Description: Template string used when generating hostnames for clients. Supported tokens: `${ip}` (IP address), `${ip.dashed}` (IP address with hyphens instead of dots/colons), `${ip.hex}` (hexadecimal IP), `${mac}` (MAC address in hex), `${mac.dashed}` (MAC address with hyphens), `${duid}` (client DUID in hex), and `${filter.name}` (name of matching filter).
- Hierarchy: all

Policy: hostname.scrub.allowUnderscores (Hostname Scrub Allow Underscores)

- Name: `hostname.scrub.allowUnderscores`
- Default: `false`
- Description: Controls whether underscores (`_`) should be preserved in sanitized hostnames. When `false`, underscores are converted to the replacement character (default `-`) for strict DNS RFC compliance.
- Hierarchy: all

Policy: hostname.scrub.replaceChar (Hostname Scrub Replacement Character)

- Name: `hostname.scrub.replaceChar`
- Default: `-`
- Description: Character substituted for non-compliant DNS characters during hostname

scrubbing.

- Hierarchy: all

Policy: hostname.scrub.collapseHyphens (Hostname Scrub Collapse Hyphens)

- Name: `hostname.scrub.collapseHyphens`
- Default: `true`
- Description: Controls whether consecutive hyphens (`--+`) in sanitized hostnames should be collapsed into a single hyphen (`-`).
- Hierarchy: all

Policy: hostname.scrub.stripDomains (Hostname Scrub Strip Domains)

- Name: `hostname.scrub.stripDomains`
- Default: `local,lan,home`
- Description: Comma-separated list of ad-hoc or mDNS domain suffixes to strip from client-supplied hostnames during sanitization.
- Hierarchy: all

Policy: hostname.sendOption12 (Hostname Send Option 12)

- Name: `hostname.sendOption12`
- Default: `on-change`
- Description: Controls when DHCPv4 Option 12 (Host Name) is returned in DHCPOFFER and DHCPACK messages. Supported values: `on-change` (return Option 12 when hostname was altered or generated), `always`, `never`, and `if-requested` (only if client requested Option 12 in Parameter Request List).
- Hierarchy: all

DHCP Protocol Policies

The DHCP protocol handler follows IETF standards. However, in test labs or specialized environments, it may be desirable to modify certain behavior. The following *expert* policies can be adjusted if necessary.

Policy: dhcp.ignoreLoopback (Ignore Loopback)

- Name: `dhcp.ignoreLoopback`
- Default: `true`
- Description: Configures whether the DHCP server ignores loopback network interfaces when binding sockets during server startup.
- Hierarchy: global

Policy: dhcp.ignoreLinkLocal (Ignore Link Local)

- Name: `dhcp.ignoreLinkLocal`
- Default: `true`
- Description: Configures whether the DHCP server ignores IPv4 link-local (169.254.0.0/16) and IPv6 link-local (fe80::/10) addresses when binding sockets during server startup.
- Hierarchy: global

Policy: dhcp.ignoreSelfPackets (Ignore Self Packets)

- Name: `dhcp.ignoreSelfPackets`
- Default: `true`
- Description: Configures whether the DHCP server discards packets received from one of the server's own local IP addresses to prevent feedback loops.
- Hierarchy: global

Policy: dhcp.processor.recentMessageTimer (Recent Message Timer)

- Name: `dhcp.processor.recentMessageTimer`
- Default: `5000`
- Description: Number of milliseconds to retain transaction state for recently processed client messages to mitigate replay attacks and duplicate packet processing (DoS protection).
- Hierarchy: global

Policy: dhcp.sendRequestedOptionsOnly (Send Requested Options Only)

- Name: `dhcp.sendRequestedOptionsOnly`
- Default: `false`
- Description: Flag indicating whether the DHCP server returns only options explicitly requested by the client in its Option Request Option / Parameter Request List, or includes all configured options.
- Hierarchy: all

Policy: dhcp.supportRapidCommit (Support Rapid Commit)

- Name: `dhcp.supportRapidCommit`
- Default: `false`
- Description: Flag indicating whether the DHCP server honors 2-message DHCPv6 Rapid Commit exchanges (SOLICIT → REPLY, RFC 8415 / RFC 3315).
- Hierarchy: global, filter, link, linkFilter

DHCPv4 Policies

The following policies configure DHCPv4-specific behaviors and filtering.

Policy: v4.header.sname (Header: Server Name)

- Name: `v4.header.sname`
- Description: Populates the optional `sname` (server host name) field in the fixed DHCPv4 message header. Typically used in network boot environments (PXE/TFTP). See also `v4TftpServerNameOption` (Option 66).
- Hierarchy: all

Policy: v4.header.filename (Header: Bootfile Name)

- Name: `v4.header.filename`
- Description: Populates the optional `file` (boot file name) field in the fixed DHCPv4 message header. Specifies the boot image file path which the client retrieves from the TFTP server. See also `v4BootFileNameOption` (Option 67).
- Hierarchy: all

Policy: v4.ignoredMacAddrs (Ignored MAC Addresses)

- Name: `v4.ignoredMacAddrs`
- Default: `000000000000, FFFFFFFFFFFF`
- Description: Comma-separated list of client MAC addresses that the DHCPv4 server will silently ignore.
- Hierarchy: all

Policy: v4.excludedMacAddrsFile (Excluded MAC Addresses File)

- Name: `v4.excludedMacAddrsFile`
- Default: empty
- Description: Path to a file containing MAC addresses to ignore/exclude from DHCPv4 service (denylist). Relative paths are resolved against `$JAGORNET_DHCP_HOME/config`. Supports one MAC address per line in any standard format (`00:11:22:33:44:55`, `00-11-22-33-44-55`, `0011.2233.4455`, `001122334455`), with comments starting with `#`, `//`, or `;`. Automatically hot-reloaded on file modification.
- Hierarchy: global

Policy: v4.includedMacAddrsFile (Included MAC Addresses File)

- Name: `v4.includedMacAddrsFile`
- Default: empty

- Description: Path to a file containing MAC addresses permitted to receive DHCPv4 service (allowlist). When populated, only clients with MAC addresses in this file receive service; all other requests are dropped. If an address appears in both allowlist and denylist files, exclusion takes precedence. Automatically hot-reloaded on file modification.
- Hierarchy: global

Policy: v4.defaultLeasetime (Default Lease Time)

- Name: `v4.defaultLeasetime`
- Default: `3600`
- Description: Default lease duration in seconds granted to DHCPv4 clients when no client lease time is requested (Option 51).
- Hierarchy: all

Policy: v4.pingCheckTimeout (Ping Check Timeout)

- Name: `v4.pingCheckTimeout`
- Default: `0`
- Description: Number of milliseconds to wait for an ICMP ping response before offering an unassigned IPv4 address to a client. When set to `0` (default), ping check is disabled.
- Hierarchy: global

DHCPv6 Policies

The following policies configure DHCPv6-specific behaviors and filtering.

Policy: v6.excludedDuidsFile (Excluded DUIDs File)

- Name: `v6.excludedDuidsFile`
- Default: empty
- Description: Path to a file containing DHCP Unique Identifiers (DUIDs) to ignore/exclude from DHCPv6 service (denylist). Relative paths are resolved against `$JAGORNET_DHCP_HOME/config`. Supports one hex DUID per line with or without colons/dashes/dots, and comment lines starting with `#`, `//`, or `;`. Automatically hot-reloaded on file modification.
- Hierarchy: global

Policy: v6.includedDuidsFile (Included DUIDs File)

- Name: `v6.includedDuidsFile`
- Default: empty
- Description: Path to a file containing client DUIDs permitted to receive DHCPv6 service (allowlist). When populated, only clients with matching DUIDs receive service; all other requests are dropped. If a DUID appears in both allowlist and denylist files, exclusion takes precedence.

Automatically hot-reloaded on file modification.

- Hierarchy: global

Policy: v6.preferredLifetime (Preferred Lifetime)

- Name: `v6.preferredLifetime`
- Default: `3600`
- Description: Preferred lifetime in seconds for IA_NA addresses or IA_PD prefixes granted to DHCPv6 clients.
- Hierarchy: global, link, pool

Policy: v6.validLifetime (Valid Lifetime)

- Name: `v6.validLifetime`
- Default: `3600`
- Description: Valid lifetime in seconds for IA_NA addresses or IA_PD prefixes granted to DHCPv6 clients.
- Hierarchy: global, link, pool

Policy: v6.verifyUnknownRebind (Verify Unknown Rebind)

- Name: `v6.verifyUnknownRebind`
- Default: `false`
- Description: Controls whether the DHCPv6 server verifies that addresses in an unknown client's REBIND request are appropriate for the receiving link per RFC 8415 / RFC 3315 section 18.2.4.
- Hierarchy: global, filter, link, linkFilter

Policy: v6.iaNaT1 (IA NA T1)

- Name: `v6.iaNaT1`
- Default: `0.5`
- Description: Percentage/fraction of shortest preferred lifetime in an IA_NA used to calculate the T1 (renewal) timer returned in DHCPv6 server replies (e.g. `0.5` = 50%).
- Hierarchy: global, link

Policy: v6.iaNaT2 (IA NA T2)

- Name: `v6.iaNaT2`
- Default: `0.8`
- Description: Percentage/fraction of shortest preferred lifetime in an IA_NA used to calculate the T2 (rebind) timer returned in DHCPv6 server replies (e.g. `0.8` = 80%).
- Hierarchy: global, link

Policy: v6.iaPdT1 (IA PD T1)

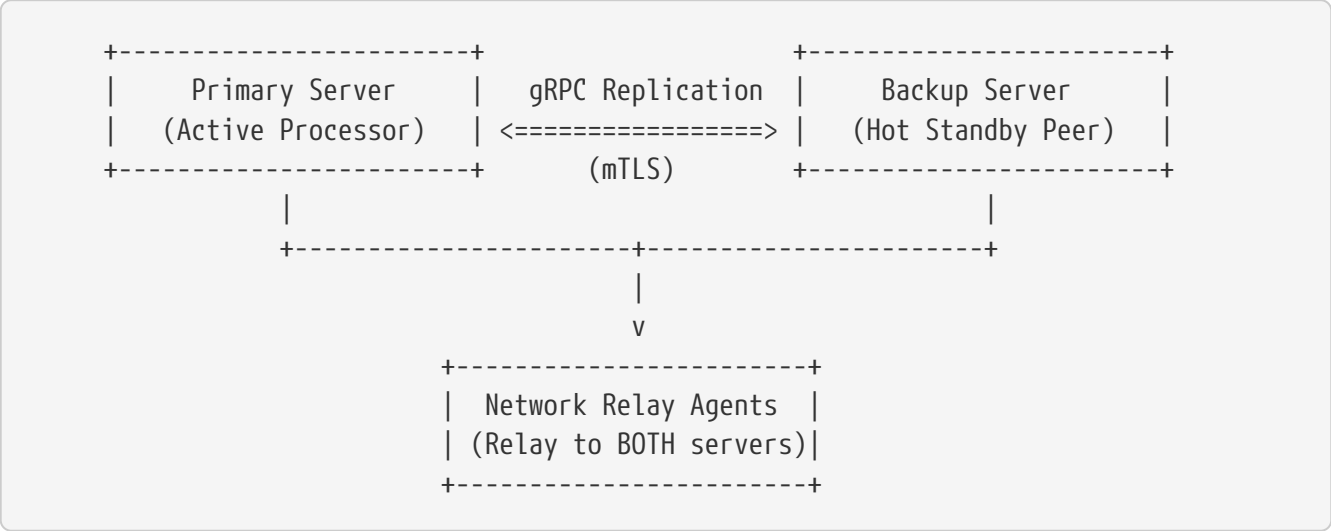
- Name: `v6.iaPdT1`
- Default: `0.5`
- Description: Percentage/fraction of shortest preferred lifetime in an IA_PD prefix delegation option used to calculate the T1 (renewal) timer returned in DHCPv6 server replies (e.g. `0.5` = 50%).
- Hierarchy: global, link

Policy: v6.iaPdT2 (IA PD T2)

- Name: `v6.iaPdT2`
- Default: `0.8`
- Description: Percentage/fraction of shortest preferred lifetime in an IA_PD prefix delegation option used to calculate the T2 (rebind) timer returned in DHCPv6 server replies (e.g. `0.8` = 80%).
- Hierarchy: global, link

High Availability

Jagornet DHCP Server provides native Active/Standby high availability designed for enterprise resilience without requiring external cluster managers (like Corosync or Pacemaker).



Operational Mechanics

- **Lease Replication via gRPC:** The Primary processes client requests and replicates lease updates to the Backup over high-speed gRPC (port `9066`) secured with Mutual TLS (mTLS).
- **Failure Detection:** The Backup continuously polls the Primary at `ha.pollSeconds` (default: 30s). If the Primary fails to respond after `ha.pollReplyFailureCount` attempts (default: 5), the Backup transitions to the active state and begins serving client leases.
- **Failback & Synchronization:** When the Primary recovers, it automatically synchronizes

changed leases from the Backup before taking over lease handling for each subnet link.

- **Router Relay Recommendation:** For seamless failover, configure DHCP relay agents on switches/routers with the IP addresses of **both** servers (e.g. `ip helper-address 10.0.0.10` and `ip helper-address 10.0.0.11`).

High Availability Configuration Example

Primary Server Configuration (`dhcpserver.yaml` on 10.0.0.10)

```
policies:
  policyList:
  - name: "ha.role"
    value: "primary"
  - name: "ha.peerServer"
    value: "10.0.0.11"
  - name: "ha.peerPort"
    value: "9066"
  - name: "ha.bindingUpdateMode"
    value: "sync"
```

Backup Server Configuration (`dhcpserver.yaml` on 10.0.0.11)

```
policies:
  policyList:
  - name: "ha.role"
    value: "backup"
  - name: "ha.peerServer"
    value: "10.0.0.10"
  - name: "ha.peerPort"
    value: "9066"
  - name: "ha.bindingUpdateMode"
    value: "sync"
```

NOTE	In the event of catastrophic failure or database corruption on the Primary, the Primary can be forced to synchronize all leases from the Backup by deleting the state database file defined by `ha.stateDbFile` before restarting.

Several policies are available to configure High Availability (HA) behavior.

Policy: ha.role (HA Role)

- Name: `ha.role`
- Description: High Availability (HA) node role:
  - `primary` - active server processing client requests and replicating to peer
  - `backup` - standby server monitoring primary and ready for hot failover
- Hierarchy: global

Policy: ha.protocol (HA Protocol)

- Name: `ha.protocol`
- Default: `grpc`
- Description: Inter-node communication protocol used for High Availability lease synchronization and heartbeat polling:
  - `grpc` - high-speed gRPC over TLS using Mutual TLS (mTLS) certificates (recommended)
  - `rest` - legacy HTTP REST endpoints using username/password basic authentication
- Hierarchy: global

Policy: ha.bindingUpdateMode (HA Binding Update Mode)

- Name: `ha.bindingUpdateMode`
- Default: `sync`
- Description: The High Availability lease update synchronization mode:
  - `sync`: synchronous - update the peer before responding to the client
  - `async`: asynchronous - update the peer in the background while responding immediately to the client
  - `database`: delegate binding updates to database cluster/replication technology
- Hierarchy: global

Policy: ha.peerServer (HA Peer Server)

- Name: `ha.peerServer`
- Description: The IP address or hostname of the remote High Availability peer server.
- Hierarchy: global

Policy: ha.peerPort (HA Peer Port)

- Name: `ha.peerPort`
- Default: `9066`
- Description: The TCP port of the remote High Availability peer server for heartbeat and lease replication traffic.
- Hierarchy: global

Policy: ha.pollSeconds (HA Poll Seconds)

- Name: `ha.pollSeconds`
- Default: `30`
- Description: Interval in seconds between periodic heartbeat poll requests sent to the High Availability peer server.

- Hierarchy: global

Policy: ha.pollReplyTimeout (HA Poll Reply Timeout)

- Name: `ha.pollReplyTimeout`
- Default: `1000`
- Description: Timeout in milliseconds to wait for a heartbeat poll reply from the High Availability peer server before recording a failure.
- Hierarchy: global

Policy: ha.pollReplyFailureCount (HA Poll Reply Failure Count)

- Name: `ha.pollReplyFailureCount`
- Default: `5`
- Description: Number of consecutive unacknowledged heartbeat poll attempts before declaring the High Availability peer server unavailable and triggering failover.
- Hierarchy: global

Policy: ha.requestAllLeasesOnRestart (HA Request All Leases On Restart)

- Name: `ha.requestAllLeasesOnRestart`
- Default: `true`
- Description: Flag indicating whether a recovering High Availability server requests a complete synchronization of all leases on restart, or only leases changed since the last communication with the peer server.
- Hierarchy: global

Policy: ha.username (HA Username)

- Name: `ha.username`
- Default: `hapeer`
- Description: Username accepted for inbound High Availability REST authentication when `ha.protocol` is set to `rest`. Ensure that the `ha.peerUsername` configured on the peer server matches this value.
- Hierarchy: global

Policy: ha.password (HA Password)

- Name: `ha.password`
- Default: `jagornet`
- Description: Password accepted for inbound High Availability REST authentication when `ha.protocol` is set to `rest`. Ensure that the `ha.peerPassword` configured on the peer server matches this value.

- Hierarchy: global

Policy: ha.peerUsername (HA Peer Username)

- Name: `ha.peerUsername`
- Default: `hapeer`
- Description: Username sent for outbound High Availability REST authentication when `ha.protocol` is set to `rest`. Ensure that this value matches the `ha.username` configured on the peer server.
- Hierarchy: global

Policy: ha.peerPassword (HA Peer Password)

- Name: `ha.peerPassword`
- Default: `jagornet`
- Description: Password sent for outbound High Availability REST authentication when `ha.protocol` is set to `rest`. Ensure that this value matches the `ha.password` configured on the peer server.
- Hierarchy: global

Policy: ha.stateDbFile (HA State DB File)

- Name: `ha.stateDbFile`
- Default: `db/ha/jagornet-ha-state.db`
- Description: File path to the local embedded database used by High Availability to track peer replication states and lease synchronization checkpoints.
- Hierarchy: global

Policy: ha.maxStoredStates (HA Max Stored States)

- Name: `ha.maxStoredStates`
- Default: `10`
- Description: Maximum number of previous synchronization checkpoints maintained in the local HA state database file.
- Hierarchy: global

REST API

The REST API is enabled by default over HTTPS port 9067 on the server host. See startup options for controlling the port or network interfaces used for HTTPS. The API is hosted at `https://<server-ip>:9067`.

NOTE	The `/dhcpleases` endpoints for external REST API clients are read-only (`GET`). Lease

> synchronization mutations (`PUT`, `POST`) are strictly restricted to the internal High Availability peer role (`ha.peerUsername` / `ha.peerPassword`), returning `403 Forbidden` for standard REST API users. Arbitrary lease deletion (`DELETE`) is disabled.

Explore the interactive OpenAPI 3.0 reference and download the raw specification online:

View Jagornet DHCP Server REST API Reference (raw spec: openapi.yaml)

Policy: rest.api.username (REST API Username)

- Name: `rest.api.username`
- Default: `jagornet`
- Description: Username for HTTP Basic Authentication to the server's management REST API and Prometheus `/metrics` endpoint.
- Hierarchy: global

Policy: rest.api.password (REST API Password)

- Name: `rest.api.password`
- Default: `jagornet`
- Description: Password for HTTP Basic Authentication to the server's management REST API and Prometheus `/metrics` endpoint.
- Hierarchy: global

Observability: Prometheus Metrics

Jagornet DHCP Server provides an embedded Prometheus `/metrics` endpoint over HTTPS for real-time telemetry, capacity planning, and alerting.

- **Endpoint:** `https://<server-ip>:9067/metrics`
- **Authentication:** Protected with HTTP Basic Authentication using the configured `rest.api.username` and `rest.api.password`.

Example Prometheus Scrape Configuration (`prometheus.yml`)

```
scrape_configs:
  - job_name: 'jagornet-dhcp'
    scheme: https
    tls_config:
      insecure_skip_verify: true   # Set to false if using enterprise CA certificates
    basic_auth:
      username: 'jagornet'
      password: 'your-secure-password'
    static_configs:
      - targets: ['dhcp1.example.com:9067', 'dhcp2.example.com:9067']
```

Exported Metrics Overview

- **Lease Allocation:** Active lease count, pool utilization percentage, and available addresses per subnet link.
- **Traffic Rates & Counters:** DHCPDISCOVER, DHCPOFFER, DHCPREQUEST, DHCPACK, and DHCPNAK messages.
- **DHCPv6 Counters:** SOLICIT, ADVERTISE, REQUEST, CONFIRM, RENEW, REBIND, and REPLY messages.
- **Server Health:** JVM heap memory, garbage collection duration, and Netty event loop thread performance.

DHCP Options

DHCP options are configured using the `options` element. Each option is specified by an element with a name of the option, for example `dnsServersOption`. The Jagornet DHCP server has pre-defined option definitions for the most common DHCPv4 and DHCPv6 options. Other options are easily defined and supported. See Generic Options for details.

DHCPv4 Server Identifier Option

The Server Identifier Option is required by the DHCPv4 protocol for the server to include in reply packets. The identifier is an IPv4 address which DHCPv4 clients will send unicast requests to. The `v4ServerIdOption` must be specified in the `config/dhcpserver.xml` file. The default `config/dhcpserver.xml` file supplied with the Jagornet DHCP server specifies an empty DHCPv4 server identifier option as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<dhc:dhcpServerConfig xmlns:dhc="http://jagornet.com/dhcp/xml">
    <v4ServerIdOption/>
</dhc:dhcpServerConfig>
```

Using this default configuration, the default IP address of the host will be set for the DHCPv4 server identifier by the Jagornet DHCP server upon initial startup. This will cause the `config/dhcpserver.xml` file to be rewritten with the populated `v4ServerIdOption`, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<dhc:dhcpServerConfig xmlns:dhc="http://jagornet.com/dhcp/xml">
  <v4ServerIdOption>
    <ipAddress>10.10.10.10</ipAddress>
  </v4ServerIdOption>
</dhc:dhcpServerConfig>
```

This is the recommended way to create a server identifier. Optionally, the `v4ServerIdOption` can be specified using the ipAddress element, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<dhc:dhcpServerConfig xmlns:dhc="http://jagornet.com/dhcp/xml">
    <v4ServerIdOption>
        <ipAddress>11.11.11.11<ipAddress>
    </v4ServerIdOption>
</dhc:dhcpServerConfig>
```

Whichever method is chosen to create the server identifier, it should not be changed once it has been created because this address will be used by clients when renewing their lease.

DHCPv6 Server Identifier Option

The Server Identifier Option is required by the DHCPv6 protocol for the server to include in reply packets. The `v6ServerIdOption` must be specified in the `config/dhcpserver.xml file`. The default `config/dhcpserver.xml` file supplied with the Jagornet DHCP server specifies an empty DHCPv6 server identifier option as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<dhc:dhcpServerConfig xmlns:dhc="http://jagornet.com/dhcp/xml">
    <v6ServerIdOption/>
</dhc:dhcpServerConfig>
```

Using this default configuration, a DUID-LLT, as defined by section 9.2 of RFC 3315, will be automatically generated by the Jagornet DHCP server upon initial startup. This will cause the `config/dhcpserver.xml` file to be rewritten with the generated `serverIdOption`, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<dhc:dhcpServerConfig xmlns:dhc="http://jagornet.com/dhcp/xml">
  <v6ServerIdOption>
    <opaqueData>
        <hexValue>0001000149EFC509001E52C94D49</hexValue>
    </opaqueData>
  </v6ServerIdOption>
</dhc:dhcpServerConfig>
```

This is the recommended way to create a server identifier. Optionally, the `v6ServerIdOption` can be specified using the asciiValue of an opaque data option type, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<dhc:dhcpServerConfig xmlns:dhc="http://jagornet.com/dhcp/xml">
    <v6ServerIdOption>
        <opaqueData>
            <asciiValue>Jagornet-DHCP-Server</asciiValue>
        </opaqueData>
    </v6ServerIdOption>
```

```
</dhc:dhcpServerConfig>
```

Whichever method is chosen to create the server identifier, it should not be changed once it has been created.

Configuration Options

Configuration Options are those options that can be configured for the server to return to clients in reply messages. For example, most network clients will need to know the address of one or more Domain Name System (DNS) servers.

DHCPv4 Configuration Options

Options are returned within the returned DHCPv4 reply packet.

DHCPv4 Subnet Mask Option

- Code: 1
- Name: `v4SubnetMaskOption`
- Reference: Section 3.3 of RFC 2132

Syntax:

xml

```
<v4SubnetMaskOption>
  <ipAddress>255.255.255.0</ipAddress>
</v4SubnetMaskOption>
```

json

```
{
  "v4SubnetMaskOption" : {
    "ipAddress" : "255.255.255.0"
  }
}
```

yaml

```
v4SubnetMaskOption:
  ipAddress: "255.255.255.0"
```

DHCPv4 Time Offset Option

- Code: 2
- Name: `v4TimeOffsetOption`
- Reference: Section 3.4 of RFC 2132

Syntax:

xml

```
<v4TimeOffsetOption>
  <unsignedInt>5000</unsignedInt>
</v4TimeOffsetOption>
```

json

```
"v4TimeOffsetOption" : {
  "unsignedInt" : 5000
}
```

yaml

```
v4TimeOffsetOption:
  unsignedInt: 5000
```

DHCPv4 Routers Option

- Code: 3
- Name: `v4RoutersOption`
- Reference: (Section 3.5 of RFC 2132)

Syntax:

xml

```
<v4RoutersOption>
  <ipAddressList>10.0.0.1</ipAddressList>
  <ipAddressList>10.0.0.2</ipAddressList>
</v4RoutersOption>
```

json

```
"v4RoutersOption" : {
  ipAddressList: [
    "10.0.0.1",
    "10.0.0.2"
  ]
}
```

yaml

```
v4RoutersOption:
  ipAddressList:
```

```
  - "10.0.0.1"
  - "10.0.0.2"
```

DHCPv4 Time Servers Option

- Code: 4
- Name: `v4TimeServersOption`
- Reference: (Section 3.6 of RFC 2132)

Syntax:

xml

```
<v4TimeServersOption>
  <ipAddressList>10.0.0.1</ipAddressList>
  <ipAddressList>10.0.0.2</ipAddressList>
</v4TimeServersOption>
```

json

```
"v4TimeServersOption" : {
  ipAddressList: [
    "10.0.0.1",
    "10.0.0.2"
  ]
}
```

yaml

```
v4TimeServersOption:
  ipAddressList:
  - "10.0.0.1"
  - "10.0.0.2"
```

DHCPv4 Domain Servers Option

- Code: 6
- Name: `v4DomainServersOption`
- Reference: (Section 3.8 of RFC 2132)

Syntax:

xml

```
<v4DomainServersOption>
  <ipAddressList>10.0.0.1</ipAddressList>
  <ipAddressList>10.0.0.2</ipAddressList>
```

```
</v4DomainServersOption>
```

xml

```
"v4DomainServersOption" : {
  ipAddressList: [
    "10.0.0.1",
    "10.0.0.2"
  ]
}
```

yaml

```
v4DomainServersOption:
  ipAddressList:
  - "10.0.0.1"
  - "10.0.0.2"
```

DHCPv4 Domain Name Option

- Code: 15
- Name: `v4DomainNameOption`
- Reference: (Section 3.17 of RFC 2132)

Syntax:

xml

```
<v4DomainNameOption>
  <domainName>foo.com.</domainName>
</v4DomainNameOption>
```

json

```
"v4DomainNameOption" : {
  "domainName" : "foo.com."
}
```

yaml

```
v4DomainNameOption:
  domainName: "foo.com."
```

DHCPv4 Vendor Specific Option

- Code: 43

- Name: `v4VendorSpecificOption`
- Reference: (Section 8.4 of of RFC 2132)

Syntax:

xml

```
<v4VendorSpecificOption>
  <opaqueData>
    <subOptions>
      <optionDefList v4="true" code="1" name="VendorSubopt1">
        <stringOption>
          <string>VendorSpecial</string>
        </stringOption>
      </optionDefList>
      <optionDefList v4="true" code="2" name="VendorSubopt2">
        <ipAddressOption>
          <ipAddress>10.11.12.13</ipAddress>
        </ipAddressOption>
      </optionDefList>
    </subOptions>
  </opaqueData>
</v4VendorSpecificOption>
```

json

```
{
  "v4VendorSpecificOption" : {
    "opaqueData" : {
      "subOptions" : {
        "optionDefList" : [ {
          "stringOption" : {
            "string" : "VendorSpecial"
          },
          "v4" : true,
          "code" : 1,
          "name" : "VendorSubopt1"
        }, {
          "ipAddressOption" : {
            "ipAddress" : "10.11.12.13"
          },
          "v4" : true,
          "code" : 2,
          "name" : "VendorSubopt2"
        } ]
      }
    }
  }
}
```

yaml

```
v4VendorSpecificOption:
  opaqueData:
    subOptions:
      optionDefList:
      - stringOption:
          string: "VendorSpecial"
        v4: true
        code: 1
        name: "VendorSubopt1"
      - ipAddressOption:
          ipAddress: "10.11.12.13"
        v4: true
        code: 2
        name: "VendorSubopt2"
```

DHCPv4 Netbios Name Servers Option

- Code: 44
- Name: `v4NetbiosNameServersOption`
- Reference: (Section 8.5 of of RFC 2132)

Syntax:

xml

```
<v4NetbiosNameServersOption>
  <ipAddressList>10.0.0.1</ipAddressList>
  <ipAddressList>10.0.0.2</ipAddressList>
</v4NetbiosNameServersOption>
```

json

```
"v4NetbiosNameServersOption" : {
  ipAddressList: [
    "10.0.0.1",
    "10.0.0.2"
  ]
}
```

yaml

```
v4NetbiosNameServersOption:
  ipAddressList:
  - "10.0.0.1"
  - "10.0.0.2"
```

DHCPv4 Netbios Node Type Option

- Code: 46
- Name: `v4NetbiosNodeTypeOption`
- Reference: (Section 8.7 of of RFC 2132)

Syntax:

xml

```
<v4NetbiosNodeTypeOption>
  <unsignedByte>8</unsignedByte>
</v4NetbiosNodeTypeOption>
```

json

```
"v4NetbiosNodeTypeOption" : {
  "unsignedByte" : 8
}
```

yaml

```
v4NetbiosNodeTypeOption:
  unsignedByte: 8
```

DHCPv4 TFTP Server Name Option

- Code: 66
- Name: `v4TftpServerNameOption`
- Reference: (Section 9.4 of of RFC 2132)

Syntax:

xml

```
<v4TftpServerNameOption>
  <string>tftp.foo.com.</string>
</v4TftpServerNameOption>
```

json

```
"v4TftpServerNameOption" : {
  "string" : "tftp.foo.com."
}
```

yaml

```
v4TftpServerNameOption:
  string: "tftp.foo.com."
```

DHCPv4 Bootfile Name Option

- Code: 67
- Name: `v4BootFileNameOption`
- Reference: (Section 9.5 of of RFC 2132)

Syntax:

xml

```
<v4BootFileNameOption>
  <string>bootfile-name</string>
</v4BootFileNameOption>
```

json

```
"v4BootFileNameOption" : {
  "string" : "bootfile-name"
}
```

yaml

```
v4BootFileNameOption:
  string: "bootfile-name"
```

DHCPv6 Configuration Options

Options can be returned at three distinct "levels" within the returned DHCPv6 reply packet.

`v6MsgConfigOptions` - Message configuration options will be returned to the client at the outermost layer of the DHCPv6 packet. For Info-Request messages, only message configuration options are returned to the client. All known configuration options are returned to the client at the message level.

`v6IaNaConfigOptions/v6IaTaConfigOptions/v6IaPdConfigOptions` - Identity association configuration options will be returned to the client inside the Identity Association (IA) option within the reply message. Separate configuration options elements are available for each type of IA option, including IA_NA, IA_TA, and IA_PD options. No known configuration options are returned to the client at the IA level, therefore these elements are for experimental and future use.

`v6NaAddrConfigOptions/v6TaAddrConfigOptions/v6PrefixConfigOptions` - Address configuration options will be returned to the client inside the address or prefix option within the IA option within the reply message. Separate configuration options elements are available for each of the associated

IA option type. No known configuration options are returned to the client at the address level, therefore these elements are for experimental and future use.

DHCPv6 Preference Option

- Code: 7
- Name: `v6PreferenceOption`
- Reference: (Section 22.8 of RFC 3315)

Syntax:

xml

```
<v6PreferenceOption>
  <unsignedByte>10</unsignedByte>
</v6PreferenceOption>
```

json

```
"v6PreferenceOption" : {
  "unsignedByte" : 10
}
```

yaml

```
v6PreferenceOption:
  unsignedByte: 10
```

DHCPv6 Server Unicast Option

- Code: 12
- Name: `v6ServerUnicastOption`
- Reference: (Section 22.8 of RFC 3315)

Syntax:

xml

```
<v6ServerUnicastOption>
  <ipAddress>2001:db8::1</ipAddress>
</v6ServerUnicastOption>
```

json

```
"v6ServerUnicastOption" : {
  "ipAddress" : "2001:db8::1"
}
```

yaml

```
v6ServerUnicastOption:
  ipAddress: "2001:db8::1"
```

DHCPv6 Status Code Option

- Code: 13
- Name: `v6StatusCodeOption`
- Reference: (Section 22.13 of RFC 3315)

Syntax:

xml

```
<v6StatusCodeOption>
  <code>5</code>
  <message>UseMulticast</message>
</v6StatusCodeOption>
```

json

```
"v6StatusCodeOption" : {
  "code" : 5,
  "message" : "UseMulticast"
}
```

yaml

```
v6StatusCodeOption:
  code: 5
  message: "UseMulticast"
```

DHCPv6 Vendor Information Option

- Code: 17
- Name: `v6VendorInfoOption`
- Reference: (Section 22.16 of RFC 3315)

Syntax:

xml

```
<v6VendorInfoOption>
  <enterpriseNumber>999</enterpriseNumber>
  <subOptions>
    <optionDefList code="1" name="VendorSubopt1">
```

```
      <stringOption>
        <string>VendorSpecial</string>
      </stringOption>
    </optionDefList>
    <optionDefList code="2" name="VendorSubopt2">
      <ipAddressOption>
        <ipAddress>2001:db8::999</ipAddress>
      </ipAddressOption>
    </optionDefList>
  </subOptions>
</v6VendorInfoOption>
```

json

```
{
  "v6VendorInfoOption" : {
    "enterpriseNumber" : 999,
    "subOptions" : {
      "optionDefList" : [ {
        "stringOption" : {
          "string" : "VendorSpecial"
        },
        "code" : 1,
        "name" : "VendorSubopt1"
      }, {
        "ipAddressOption" : {
          "ipAddress" : "2001:db8::999"
        },
        "code" : 2,
        "name" : "VendorSubopt2"
      } ]
    }
  }
}
```

yaml

```
v6VendorInfoOption:
  enterpriseNumber: 999
  subOptions:
    optionDefList:
    - stringOption:
        string: "VendorSpecial"
      code: 1
      name: "VendorSubopt1"
    - ipAddressOption:
        ipAddress: "2001:db8::999"
      code: 2
      name: "VendorSubopt2"
```

DHCPv6 SIP Server Domain Names Option

- Code: 21
- Name: `v6SipServerDomainNamesOption`
- Reference: (RFC 3319)

Syntax:

xml

```
<v6SipServerDomainNamesOption>
  <domainNameList>sip.foo.com.</domainNameList>
  <domainNameList>sip.bar.com.</domainNameList>
</v6SipServerDomainNamesOption>
```

json

```
{
  "v6SipServerDomainNamesOption" : {
    "domainNameList" : [ "sip.foo.com.", "sip.bar.com." ]
  }
}
```

yaml

```
v6SipServerDomainNamesOption:
  domainNameList:
  - "sip.foo.com."
  - "sip.bar.com."
```

DHCPv6 SIP Server Adresses Option

- Code: 22
- Name: `v6SipServerAddressesOption`
- Reference: (RFC 3319)

Syntax:

xml

```
<v6SipServerAddressesOption>
  <ipAddressList>2001:db8::1</ipAddressList>
  <ipAddressList>2001:db8::2</ipAddressList>
</v6SipServerAddressesOption>
```

json

```
{
```

```
  "v6SipServerAddressesOption" : {
    "ipAddressList" : [ "2001:db8::1", "2001:db8::2" ]
  }
}
```

yaml

```
v6SipServerAddressesOption:
  ipAddressList:
  - "2001:db8::1"
  - "2001:db8::2"
```

DHCPv6 DNS Servers Option

- Code: 23
- Name: `v6DnsServersOption`
- Reference: (RFC 3646)

Syntax:

xml

```
<v6DnsServersOption>
  <ipAddressList>2001:db8::1</ipAddressList>
  <ipAddressList>2001:db8::2</ipAddressList>
</v6DnsServersOption>
```

json

```
{
  "v6SipServerAddressesOption" : {
    "ipAddressList" : [ "2001:db8::1", "2001:db8::2" ]
  }
}
```

yaml

```
v6SipServerAddressesOption:
  ipAddressList:
  - "2001:db8::1"
  - "2001:db8::2"
```

DHCPv6 Domain Search Option

- Code: 24
- Name: `v6DomainSearchListOption`

- Reference: (RFC 3646)

Syntax:

xml

```
<v6DomainSearchListOption>
  <domainNameList>foo.com.</domainNameList>
  <domainNameList>bar.com.</domainNameList>
</v6DomainSearchListOption>
```

json

```
{
  "v6DomainSearchListOption" : {
    "domainNameList" : [ "foo.com.", "bar.com." ],
  }
}
```

yaml

```
name: "DhcpV6DomainSearchListOption"
  domainNameList:
  - "foo.com."
  - "bar.com."
```

DHCPv6 NIS Servers Option

- Code: 27
- Name: `v6NisServersOption`
- Reference: (RFC 3898)

Syntax:

xml

```
<v6NisServersOption>
  <ipAddressList>2001:db8::1</ipAddressList>
  <ipAddressList>2001:db8::2</ipAddressList>
</v6NisServersOption>
```

json

```
{
  "v6NisServersOption" : {
    "ipAddressList" : [ "2001:db8::1", "2001:db8::2" ]
  }
```

```
}
```

yaml

```
v6NisServersOption:
  ipAddressList:
  - "2001:db8::1"
  - "2001:db8::2"
```

DHCPv6 NIS+ Servers Option

- Code: 28
- Name: `v6NisPlusServersOption`
- Reference: (RFC 3898)

Syntax:

xml

```
<v6NisPlusServersOption>
  <ipAddressList>2001:db8::1</ipAddressList>
  <ipAddressList>2001:db8::2</ipAddressList>
</v6NisPlusServersOption>
```

json

```
{
  "v6NisServersOption" : {
    "ipAddressList" : [ "2001:db8::1", "2001:db8::2" ]
  }
}
```

yaml

```
v6NisServersOption:
  ipAddressList:
  - "2001:db8::1"
  - "2001:db8::2"
```

DHCPv6 NIS Domain Name Option

- Code: 29
- Name: `v6NisDomainNameOption`
- Reference: (RFC 3898)

Syntax:

xml

```
<v6NisDomainNameOption>
  <domainName>foo.com.</domainName>
</v6NisDomainNameOption>
```

json

```
{
  "v6NisDomainNameOption" : {
    "domainName" : "foo.com."
  }
}
```

yaml

```
v6NisDomainNameOption:
  domainName: "foo.com."
```

DHCPv6 NIS+ Domain Name Option

- Code: 30
- Name: `v6NisPlusDomainNameOption`
- Reference: (RFC 3898)

Syntax:

xml

```
<v6NisPlusDomainNameOption>
  <domainName>foo.com.</domainName>
</v6NisPlusDomainNameOption>
```

json

```
{
  "v6NisDomainNameOption" : {
    "domainName" : "foo.com."
  }
}
```

yaml

```
v6NisDomainNameOption:
  domainName: "foo.com."
```

DHCPv6 SNTP Servers Option

- Code: 31
- Name: `v6SntpServersOption`
- Reference: (RFC 4075)

Syntax:

xml

```
<v6SntpServersOption>
  <ipAddressList>2001:db8::1</ipAddressList>
  <ipAddressList>2001:db8::2</ipAddressList>
</v6SntpServersOption>
```

json

```
{
  "v6SntpServersOption" : {
    "ipAddressList" : [ "2001:db8::1", "2001:db8::2" ]
  }
}
```

yaml

```
v6SntpServersOption:
  ipAddressList:
  - "2001:db8::1"
  - "2001:db8::2"
```

DHCPv6 Info Refresh Time Option

- Code: 32
- Name: `v6InfoRefreshTimeOption`
- Reference: (RFC 4242)

Syntax:

xml

```
<v6InfoRefreshTimeOption>
  <unsignedInt>3600</unsignedInt>
</v6InfoRefreshTimeOption>
```

json

```
{
  "v6InfoRefreshTimeOption" : {
```

```
    "unsignedInt" : 3600
  }
}
```

yaml

```
v6InfoRefreshTimeOption:
  unsignedInt: 3600
```

DHCPv6 BCMS Domain Names Option

- Code: 33
- Name: `v6BcmcsDomainNamesOption`
- Reference: (RFC 4280)

Syntax:

xml

```
<v6BcmcsDomainNamesOption>
  <domainNameList>bcmcs.foo.com.</domainNameList>
  <domainNameList>bcmcs.bar.com.</domainNameList>
</v6BcmcsDomainNamesOption>
```

json

```
{
  "v6BcmcsDomainNamesOption" : {
    "domainNameList" : [ "bcmcs.foo.com.", "bcmcs.bar.com." ]
  }
}
```

yaml

```
v6BcmcsDomainNamesOption:
  domainNameList:
  - "bcmcs.foo.com."
  - "bcmcs.bar.com."
```

DHCPv6 BCMS Adresses Option

- Code: 34
- Name: `v6BcmcsAddressesOption`
- Reference: (RFC 4280)

Syntax:

xml

```
<v6BcmcsAddressesOption>
  <ipAddressList>2001:db8::1</ipAddressList>
  <ipAddressList>2001:db8::2</ipAddressList>
</v6BcmcsAddressesOption>
```

json

```
{
  "v6BcmcsAddressesOption" : {
    "ipAddressList" : [ "2001:db8::1", "2001:db8::2" ]
  }
}
```

yaml

```
v6BcmcsAddressesOption:
  ipAddressList:
  - "2001:db8::1"
  - "2001:db8::2"
```

DHCPv6 Geoconf Civic Option

- Code: 36
- Name: `v6GeoconfCivicOption`
- Reference: (RFC 4776)

Syntax:

xml

```
<v6GeoconfCivicOption>
  <what>1</what>
  <countryCode>US</countryCode>
  <civicAddressElementList>
    <caType>0</caType>
    <caValue>de</caValue>
  </civicAddressElementList>
  <civicAddressElementList>
    <caType>128</caType>
    <caValue>Latn</caValue>
  </civicAddressElementList>
  <civicAddressElementList>
    <caType>1</caType>
    <caValue>Bayern</caValue>
  </civicAddressElementList>
</v6GeoconfCivicOption>
```

json

```
{
  "v6GeoconfCivicOption" : {
    "what" : 1,
    "countryCode" : "US",
    "civicAddressElementList" : [ {
      "caType" : 0,
      "caValue" : "de"
    }, {
      "caType" : 128,
      "caValue" : "Latn"
    }, {
      "caType" : 1,
      "caValue" : "Bayern"
    } ]
  }
}
```

yaml

```
v6GeoconfCivicOption:
  what: 1
  countryCode: "US"
  civicAddressElementList:
  - caType: 0
    caValue: "de"
  - caType: 128
    caValue: "Latn"
  - caType: 1
    caValue: "Bayern"
```

DHCPv6 PANA Agent Addresses Option

- Code: 40
- Name: `v6PanaAgentAddressesOption`
- Reference: (RFC 5192)

Syntax:

xml

```
<v6PanaAgentAddressesOption>
  <ipAddressList>2001:db8::1</ipAddressList>
  <ipAddressList>2001:db8::2</ipAddressList>
</v6PanaAgentAddressesOption>
```

json

```
{
  "v6PanaAgentAddressesOption" : {
    "ipAddressList" : [ "2001:db8::1", "2001:db8::2" ]
  }
}
```

yaml

```
v6PanaAgentAddressesOption:
  ipAddressList:
  - "2001:db8::1"
  - "2001:db8::2"
```

DHCPv6 New Posix Timezone Option

- Code: 41
- Name: `v6NewPosixTimezoneOption`
- Reference: (RFC 4833)

Syntax:

xml

```
<v6NewPosixTimezoneOption>
  <string>EST5EDT4,M3.2.0/02:00,M11.1.0/02:00</string>
</v6NewPosixTimezoneOption>
```

json

```
{
  "v6NewPosixTimezoneOption" : {
    "string" : "EST5EDT4,M3.2.0/02:00,M11.1.0/02:00"
  }
}
```

yaml

```
v6NewPosixTimezoneOption:
  string: "EST5EDT4,M3.2.0/02:00,M11.1.0/02:00"
```

DHCPv6 New TZ DB Timezone Option

- Code: 42
- Name: `v6NewTzdbTimezoneOption`

- Reference: (RFC 4833)

Syntax:

xml

```
<v6NewTzdbTimezoneOption>
  <string>Europe/Zurich</string>
</v6NewTzdbTimezoneOption>
```

json

```
{
  "v6NewTzdbTimezoneOption" : {
    "string" : "Europe/Zurich"
  }
}
```

yaml

```
v6NewTzdbTimezoneOption:
  string: "Europe/Zurich"
```

DHCPv6 Lost Server Domain Name Option

- Code: 51
- Name: `v6LostServerDomainNameOption`
- Reference: (RFC 4253)

Syntax:

xml

```
<v6LostServerDomainNameOption>
  <domainName>lost.foo.com.</domainName>
</v6LostServerDomainNameOption>
```

json

```
{
  "v6LostServerDomainNameOption" : {
    "domainName" : "lost.foo.com."
  }
}
```

yaml

```
v6LostServerDomainNameOption:
```

```
  domainName: "lost.foo.com."
```

Generic Options

Generic options are used to define new option types for experimental, future or any standard options that are not defined above for DHCPv4 and DHCPv6 configuration options. Generic options are defined using `optionDefList` elements when specifying `v4OtherOptions` or `v6OtherOptions` element of the configuration options can be used to add one or more undefined options to the options that will be sent by the server to the client. Generic options are also used when defining the `subOptions` of the DHCPv4 and DHCPv6 Vendor Information Options as shown in the table above.

Option Definition Type

The option definition type predefines several option types for use in creating new, experimental, or vendor options.

NOTE	The option definition type examples in the table below are shown for the `subOptions` element, but the syntax is also used for the `v4OtherOptions` and `v6OtherOptions` elements.

IMPORTANT	DHCPv4 option definitions must include the `v4="true"` attribute.

Nil Option

- Element: `nilOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <nilOption/>
  </optionDefList>
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
      "nilOption" : { },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - nilOption: {}
    code: 99
    name: "MyOption"
```

Unsigned Byte Option

- Element: `uByteOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <uByteOption>
      <unsignedByte>255</unsignedByte>
    </uByteOption>
  </optionDefList>
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
      "uByteOption" : {
        "unsignedByte" : 255
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - uByteOption:
      unsignedByte: 255
    code: 99
    name: "MyOption"
```

Unsigned Byte List Option

- Element: `uByteListOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <uByteListOption>
      <unsignedByteList>1</unsignedByteList>
      <unsignedByteList>10</unsignedByteList>
      <unsignedByteList>255</unsignedByteList>
    </uByteListOption>
  </optionDefList>
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
      "uByteListOption" : {
        "unsignedByteList" : [ 1, 10, 255 ]
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - uByteListOption:
      unsignedByteList:
      - 1
      - 10
      - 255
    code: 99
    name: "MyOption"
```

Unsigned Short Option

- Element: `uShortOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <uShortOption>
      <unsignedShort>65535</unsignedShort>
    </uShortOption>
  </optionDefList>
</subOptions>
```

json

```
{
  "genericOptionsType" : {
    "optionDefList" : [ {
      "uShortOption" : {
        "unsignedShort" : 65535
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
genericOptionsType:
  optionDefList:
  - uShortOption:
      unsignedShort: 65535
    code: 99
    name: "MyOption"
```

Unsigned Short List Option

- Element: `uShortListOption`

Syntax:

xml

```
<subOptions>
<optionDefList code="99" name="MyOption">
  <uShortListOption>
    <unsignedShortList>1</unsignedShortList>
      <unsignedShortList>999</unsignedShortList>
      <unsignedShortList>65535</unsignedShortList>
    </uShortListOption>
  </optionDefList>
```

```
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
      "uShortListOption" : {
        "unsignedShortList" : [ 1, 999, 65535 ]
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - uShortListOption:
      unsignedShortList:
      - 1
      - 999
      - 65535
    code: 99
    name: "MyOption"
```

Unsigned Integer Option

- Element: `uIntOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <uIntOption>
      <unsignedInt>4294697295</unsignedInt>
    </uIntOption>
  </optionDefList>
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
```

```
      "uIntOption" : {
        "unsignedInt" : 4294697295
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - uIntOption:
      unsignedInt: 4294697295
    code: 99
    name: "MyOption"
```

String Option

- Element: `stringOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <stringOption>
      <string>myOptionStringValue</string>
    </stringOption>
  </optionDefList>
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
      "stringOption" : {
        "string" : "myOptionStringValue"
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - stringOption:
      string: "myOptionStringValue"
    code: 99
    name: "MyOption"
```

IP Address Option

- Element: `ipAddressOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <ipAddressOption>
      <ipAddress>2001:db8::1</ipAddress>
    </ipAddressOption>
  </optionDefList>
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
      "ipAddressOption" : {
        "ipAddress" : "2001:db8::1"
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - ipAddressOption:
      ipAddress: "2001:db8::1"
    code: 99
    name: "MyOption"
```

IP Address List Option

- Element: `ipAddressListOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <ipAddressListOption>
      <ipAddressList>2001:db8::1</ipAddressList>
      <ipAddressList>2001:db8::2</ipAddressList>
      <ipAddressList>2001:db8::3</ipAddressList>
    </ipAddressListOption>
  </optionDefList>
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
      "ipAddressListOption" : {
        "ipAddressList" : [ "2001:db8::1", "2001:db8::2", "2001:db8::3" ]
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - ipAddressListOption:
      ipAddressList:
      - "2001:db8::1"
      - "2001:db8::2"
      - "2001:db8::3"
    code: 99
    name: "MyOption"
```

Domain Name Option

- Element: `domainNameOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <domainNameOption>
      <domainName>my.foo.com.</domainName>
    </domainNameOption>
  </optionDefList>
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
      "domainNameOption" : {
        "domainName" : "my.foo.com."
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - domainNameOption:
      domainName: "my.foo.com."
    code: 99
    name: "MyOption"
```

Domain Name List Option

- Element: `domainNameListOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <domainNameListOption>
      <domainNameList>my.foo.com.</domainNameList>
      <domainNameList>my.bar.com.</domainNameList>
      <domainNameList>my.yuk.com.</domainNameList>
    </domainNameListOption>
  </optionDefList>
```

```
</subOptions>
```

json

```
{
  "subOptions" : {
    "optionDefList" : [ {
      "domainNameListOption" : {
        "domainNameList" : [ "my.foo.com.", "my.bar.com.", "my.yuk.com." ]
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - domainNameListOption:
      domainNameList:
      - "my.foo.com."
      - "my.bar.com."
      - "my.yuk.com."
    code: 99
    name: "MyOption"
```

Opaque Data Option

- Element: `opaqueDataOption`

Syntax:

xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <opaqueDataOption>
      <opaqueData>
        <hexValue>0123456789abcdef</hexValue>
      </opaqueData>
    </opaqueDataOption>
  </optionDefList>
</subOptions>
```

json

```
{
```

```
  "subOptions" : {
    "optionDefList" : [ {
      "opaqueDataOption" : {
        "opaqueData" : {
          "hexValue" : "0123456789ABCDEF"
        }
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - opaqueDataOption:
      opaqueData:
        hexValue: "0123456789ABCDEF"
    code: 99
    name: "MyOption"
```

Opaque Data List Option

- Element: `opaqueDataListOption`

Syntax: .xml

```
<subOptions>
  <optionDefList code="99" name="MyOption">
    <opaqueDataListOption>
      <opaqueDataList>
        <hexValue>0123456789abcdef</hexValue>
      </opaqueDataList>
      <opaqueDataList>
        <asciiValue>HelloWorld</asciiValue>
      </opaqueDataList>
      <opaqueDataList>
        <hexValue>0a1b2c3d4e5f</hexValue>
      </opaqueDataList>
    </opaqueDataListOption>
  </optionDefList>
</subOptions>
```

json

```
{
  "subOptions" : {
```

```
    "optionDefList" : [ {
      "opaqueDataListOption" : {
        "opaqueDataList" : [ {
          "hexValue" : "0123456789ABCDEF"
        }, {
          "asciiValue" : "HelloWorld"
        }, {
          "hexValue" : "0A1B2C3D4E5F"
        } ]
      },
      "code" : 99,
      "name" : "MyOption"
    } ]
  }
}
```

yaml

```
subOptions:
  optionDefList:
  - opaqueDataListOption:
      opaqueDataList:
      - hexValue: "0123456789ABCDEF"
      - asciiValue: "HelloWorld"
      - hexValue: "0A1B2C3D4E5F"
    code: 99
    name: "MyOption"
```

Opaque Data Option

Opaque data options are those options which can contain opaque, binary data. Often, these options actually contain simple ASCII strings. Therefore, the `opaqueData` element contains either a `hexValue` element, which specifies the binary value as a hexadecimal string, or an `asciiValue` element, which specifies the ASCII string value.

Filters

Filters are used to classify DHCP clients so that specific configuration options can be supplied to certain classes of clients. A typical use of filters is to define a vendor class mapping to provide vendor specific information option data for clients that include the vendor class option in the request. Filters can also be used to arbitrarily group clients according to any criteria which matches one or more options supplied by the client. Each filter definition includes a name, one or more filter expressions, a set of one or more configured options, and optional server policies.

Filter Expressions

If more than one filter expression is defined in a filter, then the client request must match *all* of the expressions. That is, multiple filter expressions are logically *anded* together to form the match

criteria. Each filter expression must contain at least one client class, option or custom expression.

Client Class Expression

A client class expression defines criteria for matching all or part of a DHCPv4 vendor class, or a DHCPv6 user or vendor class option provided in the client request. The `clientClassExpression` element must specify only one of the three supported client class options - DHCPv4 vendor class, or DHCPv6 user or vendor class - and the required `operator` attribute, which defaults to `equals` and defines the match criteria.

DHCPv4 Vendor Class Filter Example

As another example, consider the following filter definition which matches DHCPv4 clients which supply a vendor class option beginning with the specified ASCII value. This filter is configured to provide the vendor specific information option for such clients.

xml

```
<filter>
  <name>DHCPv4 VendorClass Filter</name>
  <filterExpressions>
    <filterExpressionList>
      <clientClassExpression operator="startsWith">
        <v4VendorClassOption>
          <opaqueData>
            <asciiValue>MyVendorPrefix</asciiValue>
          </opaqueData>
        </v4VendorClassOption>
      </clientClassExpression>
    </filterExpressionList>
  </filterExpressions>
  <v4ConfigOptions>
    <v4VendorSpecificOption>
      <opaqueData>
        <hexValue>01020304</hexValue>
      </opaqueData>
    </v4VendorSpecificOption>
  </v4ConfigOptions>
</filter>
```

json

```
{
  "filter" : {
    "name" : "DHCPv4 VendorClass Filter",
    "filterExpressions" : {
      "filterExpressionList" : [ {
        "clientClassExpression" : {
          "v4VendorClassOption" : {
            "opaqueData" : {
```

```
            "asciiValue" : "MyVendorPrefix"
          }
        },
        "operator" : "startsWith"
      }
    } ]
  },
  "v4ConfigOptions" : {
    "v4VendorSpecificOption" : {
      "opaqueData" : {
        "hexValue" : "01020304"
      }
    }
  }
}
```

yaml

```
filter:
  name: "DHCPv4 VendorClass Filter"
  filterExpressions:
    filterExpressionList:
    - clientClassExpression:
        v4VendorClassOption:
          opaqueData:
            asciiValue: "MyVendorPrefix"
        operator: "startsWith"
  v4ConfigOptions:
    v4VendorSpecificOption:
      opaqueData:
        hexValue: "01020304"
```

DHCPv6 User Class Filter Example

The following filter definition matches clients which supply a specific DHCPv6 user class option value. This filter is configured to provide a specific DNS domain name for such clients.

xml

```
<filter>
  <name>DHCPv6 UserClass Filter</name>
  <filterExpressions>
    <filterExpressionList>
      <clientClassExpression operator="equals">
        <v6UserClassOption>
          <opaqueDataList>
            <asciiValue>MyUserClass</asciiValue>
          </opaqueDataList>
        </v6UserClassOption>
```

```
      </clientClassExpression>
    </filterExpressionList>
  </filterExpressions>
  <v6MsgConfigOptions>
    <v6DomainSearchListOption>
      <domainNameList>filter.com.</domainNameList>
    </v6DomainSearchListOption>
  </v6MsgConfigOptions>
</filter>
```

json

```
{
  "filter" : {
    "name" : "DHCPv6 UserClass Filter",
    "filterExpressions" : {
      "filterExpressionList" : [ {
        "clientClassExpression" : {
          "v6UserClassOption" : {
            "opaqueDataList" : [ {
              "asciiValue" : "MyUserClass"
            } ]
          },
          "operator" : "equals"
        }
      } ]
    },
    "v6MsgConfigOptions" : {
      "v6DomainSearchListOption" : {
        "domainNameList" : [ "filter.com." ]
      }
    }
  }
}
```

yaml

```
filter:
  name: "DHCPv6 UserClass Filter"
  filterExpressions:
    filterExpressionList:
    - clientClassExpression:
        v6UserClassOption:
          opaqueDataList:
          - asciiValue: "MyUserClass"
        operator: "equals"
  v6MsgConfigOptions:
    v6DomainSearchListOption:
      domainNameList:
```

```
      - "filter.com."
```

DHCPv6 Vendor Class Filter Example

The following filter definition matches client which supply a specific DHCPv6 vendor class option value. This filter is configured to provide a vendor specific information option, which contains two suboptions, for such clients.

xml

```
<filter>
  <name>DHCPv6 VendorClass Filter</name>
  <filterExpressions>
    <filterExpressionList>
      <clientClassExpression operator="equals">
        <v6VendorClassOption>
          <opaqueDataList>
            <asciiValue>VendorXYZ</asciiValue>
          </opaqueDataList>
          <enterpriseNumber>12345</enterpriseNumber>
        </v6VendorClassOption>
      </clientClassExpression>
    </filterExpressionList>
  </filterExpressions>
  <v6MsgConfigOptions>
    <v6VendorInfoOption>
      <enterpriseNumber>12345</enterpriseNumber>
      <suboptionList>
        <optionDefList code="1">
          <stringOption>
            <string>hello</string>
          </stringOption>
        </optionDefList>
        <optionDefList code="2">
          <ipAddressOption>
            <ipAddress>2001:db8::1</ipAddress>
          </ipAddressOption>
        </optionDefList>
      </suboptionList>
    </v6VendorInfoOption>
  </v6MsgConfigOptions>
</filter>
```

json

```
{
  "filter" : {
    "name" : "DHCPv6 VendorClass Filter",
    "filterExpressions" : {
      "filterExpressionList" : [ {
```

```
        "clientClassExpression" : {
          "v6VendorClassOption" : {
            "enterpriseNumber" : 12345,
            "opaqueDataList" : [ {
              "asciiValue" : "VendorXYZ"
            } ]
          },
          "operator" : "equals"
        }
      } ]
    },
    "v6MsgConfigOptions" : {
      "v6VendorInfoOption" : {
        "enterpriseNumber" : 12345,
        "suboptionList" : {
          "optionDefList" : [ {
            "stringOption" : {
              "string" : "hello"
            },
            "code" : 1
          }, {
            "ipAddressOption" : {
              "ipAddress" : "2001:db8::1"
            },
            "code" : 2
          } ]
        }
      }
    }
  }
}
```

yaml

```
filter:
  name: "DHCPv6 VendorClass Filter"
  filterExpressions:
    filterExpressionList:
    - clientClassExpression:
        v6VendorClassOption:
          enterpriseNumber: 12345
          opaqueDataList:
          - asciiValue: "VendorXYZ"
        operator: "equals"
  v6MsgConfigOptions:
    v6VendorInfoOption:
      enterpriseNumber: 12345
      suboptionList:
        optionDefList:
        - stringOption:
```

```
            string: "hello"
          code: 1
        - ipAddressOption:
            ipAddress: "2001:db8::1"
          code: 2
```

Option Expression

An option expression defines the criteria for matching all or part of an option provided in the client request using the generic option definition, along with a value and operator, which forms the expression. The `optionExpression` element must specify the DHCPv6 option code in the `code` attribute, followed by the optional `name` attribute from the generic option definition type above, and the required `operator` attribute, which defaults to `equals`.

Filter Option Expressions

Unsigned Byte Option Expression

- Type: `uByteOption`
- Operators: `equals`, `lessThan`, `lessThanOrEqual`, `greaterThan`, `greaterThanOrEqual`

Example syntax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="equals">
      <uByteOption>
        <unsignedByte>255</unsignedByte>
      </uByteOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "uByteOption" : {
          "unsignedByte" : 255
        },
        "code" : 99,
        "operator" : "equals"
      }
    } ]
  }
```

```
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      uByteOption:
        unsignedByte: 255
      code: 99
      operator: "equals"
```

Unsigned Byte List Option Expression

- Type: `uByteListOption`
- Operators: `equals`, `contains`

Example syntax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="equals">
      <uByteListOption>
        <unsignedByteList>1</unsignedByteList>
        <unsignedByteList>255</unsignedByteList>
      </uByteListOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "uByteListOption" : {
          "unsignedByteList" : [ 1, 255 ]
        },
        "code" : 99,
        "operator" : "equals"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      uByteListOption:
        unsignedByteList:
        - 1
        - 255
      code: 99
      operator: "equals"
```

Unsigned Short Option Expression

- Type: `uShortOption`
- Operators: `equals`, `lessThan`, `lessThanOrEqual`, `greaterThan`, `greaterThanOrEqual`

Example syntax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="lessThan">
      <uShortOption>
        <unsignedShort>65535</unsignedShort>
      </uShortOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "uShortOption" : {
          "unsignedShort" : 65535
        },
        "code" : 99,
        "operator" : "lessThan"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      uShortOption:
        unsignedShort: 65535
      code: 99
      operator: "lessThan"
```

Unsigned Short Option List Expression

- Type: `uShortListOption`
- Operators: `equals`, `contains`

Example syntax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="equals">
      <uShortListOption>
        <unsignedShortList>1</unsignedShortList>
        <unsignedShortList>65535</unsignedShortList>
      </uShortListOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "uShortListOption" : {
          "unsignedShortList" : [ 1, 65535 ]
        },
        "code" : 99,
        "operator" : "equals"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
```

```
  filterExpressionList:
  - optionExpression:
      uShortListOption:
        unsignedShortList:
        - 1
        - 65535
      code: 99
      operator: "equals"
```

Unsigned Integer Option Expression

- Type: `uIntOption`
- Operators: `equals`, `lessThan`, `lessThanOrEqual`, `greaterThan`, `greaterThanOrEqual`

Example syntax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="lessThan">
      <uIntOption>
        <unsignedInt>10000</unsignedInt>
      </uIntOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "uIntOption" : {
          "unsignedInt" : 10000
        },
        "code" : 99,
        "operator" : "lessThan"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
```

```
      uIntOption:
        unsignedInt: 10000
      code: 99
      operator: "lessThan"
```

String Option Expression

- Type: `stringOption`
- Operators: `equals`, `startsWith`, `endsWith`, `contains`, `regExp`

Example syntax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="endsWith">
      <stringOption>
        <string>mySuffix</string>
      </stringOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "stringOption" : {
          "string" : "mySuffix"
        },
        "code" : 99,
        "operator" : "endsWith"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      stringOption:
        string: "mySuffix"
      code: 99
```

```
      operator: "endsWith"
```

IP Address Option Expression

- Type: `ipAddressOption`
- Operators: `equals`, `startsWith`, `endsWith`, `contains`, `regExp`

Example syntax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="equals">
      <ipAddressOption>
        <ipAddress>2001:db8::1</ipAddress>
      </ipAddressOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "ipAddressOption" : {
          "ipAddress" : "2001:db8::1"
        },
        "code" : 99,
        "operator" : "equals"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      ipAddressOption:
        ipAddress: "2001:db8::1"
      code: 99
      operator: "equals"
```

IP Address List Option Expression

- Type: `ipAddressListOption`
- Operators: `equals`, `contains`

Example sytax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="contains">
      <ipAddressListOption>
        <ipAddressList>2001:db8::1</ipAddressList>
      </ipAddressListOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "ipAddressListOption" : {
          "ipAddressList" : [ "2001:db8::1" ]
        },
        "code" : 99,
        "operator" : "contains"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      ipAddressListOption:
        ipAddressList:
        - "2001:db8::1"
      code: 99
      operator: "contains"
```

Domain Name Option Expression

- Type: `domainNameOption`

- Operators: `equals`, `startsWith`, `endsWith`, `contains`, `regExp`

Example syntax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="equals">
      <domainNameOption>
        <domainName>foo.com.</domainName>
      </domainNameOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "domainNameOption" : {
          "domainName" : "foo.com."
        },
        "code" : 99,
        "operator" : "equals"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      domainNameOption:
        domainName: "foo.com."
      code: 99
      operator: "equals"
```

Domain Name List Option Expression

- Type: `domainNameListOption`
- Operators: `equals`, `contains`

Example sytax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="contains">
      <domainNameListOption>
        <domainNameList>foo.com.</domainNameList>
      </domainNameListOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "domainNameListOption" : {
          "domainNameList" : [ "foo.com." ]
        },
        "code" : 99,
        "operator" : "contains"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      domainNameListOption:
        domainNameList:
        - "foo.com."
      code: 99
      operator: "contains"
```

Opaque Data Option Expression

- Type: `opaqueDataOption`
- Operators: `equals`, `startsWith`, `endsWith`, `contains`, `regExp`

Example syntax:

xml

```
<filterExpressions>
```

```
  <filterExpressionList>
    <optionExpression code="99" operator="regExp">
      <opaqueDataOption>
        <opaqueData>
          <asciiValue>myRegularExpression</asciiValue>
        </opaqueData>
      </opaqueDataOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "opaqueDataOption" : {
          "opaqueData" : {
            "asciiValue" : "myRegularExpression"
          }
        },
        "code" : 99,
        "operator" : "regExp"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      opaqueDataOption:
        opaqueData:
          asciiValue: "myRegularExpression"
      code: 99
      operator: "regExp"
```

Opaque Data List Option Expression

- Type: `opaqueDataListOption`
- Operators: `equals`, `contains`

Example sytax:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression code="99" operator="equals">
      <opaqueDataListOption>
        <opaqueDataList>
          <asciiValue>opaqueAsciiData</asciiValue>
        </opaqueDataList>
      </opaqueDataListOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "opaqueDataListOption" : {
          "opaqueDataList" : [ {
            "asciiValue" : "opaqueAsciiData"
          } ]
        },
        "code" : 99,
        "operator" : "equals"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      opaqueDataListOption:
        opaqueDataList:
        - asciiValue: "opaqueAsciiData"
      code: 99
      operator: "equals"
```

DHCPv4 Option Expression

DHCPv4 option expressions use the same syntax as DHCPv6 option expressions, but must identify the option as a DHCPv4 option using the `v4` attribute. For example:

xml

```
<filterExpressions>
  <filterExpressionList>
    <optionExpression v4="true" code="99" operator="equals">
      <uByteOption>
        <unsignedByte>255</unsignedByte>
      </uByteOption>
    </optionExpression>
  </filterExpressionList>
</filterExpressions>
```

json

```
{
  "filterExpressions" : {
    "filterExpressionList" : [ {
      "optionExpression" : {
        "uByteOption" : {
          "unsignedByte" : 255
        },
        "v4" : true,
        "code" : 99,
        "operator" : "equals"
      }
    } ]
  }
}
```

yaml

```
filterExpressions:
  filterExpressionList:
  - optionExpression:
      uByteOption:
        unsignedByte: 255
      v4: true
      code: 99
      operator: "equals"
```

Custom Expressions

Custom expressions are used to define filter expressions that cannot be configured using standard option expressions. Currently, Jagornet DHCP Server does not support custom expressions.

Links

A link defines a network segment for client requests. At least one DHCPv4 or DHCPv6 link is

required proper server configuration. The Jagornet DHCP Server uses the link definition to classify each incoming client request. Once the client link is determined, the server will use the link definition to determine which addresses are available from the pools and/or bindings defined within the link. Additional configuration elements can be specified within the link including filters, policies, and options.

DHCPv4 Links

For DHCPv4, the link is determined by the 'giAddr' field of the DHCPv4 request header. If the 'giAddr' field is zero, then the link is determined by the IP address assigned to the IPv4 broadcast interface provided to the server at startup.

DHCPv4 Link Example

xml

```
<link>
  <name>IPv4 Client Link 1</name>
  <!-- All DHCPv4 links are defined in CIDR notation.
       For local links, specify the interface as a
       startup command-line option, and configure the
       subnet for that interface's IPv4 address. -->
    <address>10.0.0.0/24</address>
    ...
</link>
```

json

```
{
  "link" : {
    "name" : "IPv4 Client Link 1",
    "address" : "10.0.0.0/24"
    ...
  }
}
```

yaml

```
link:
  name: "IPv4 Client Link 1"
  # All DHCPv4 links are defined in CIDR notation.
  # For local links, specify the interface as a
  # startup command-line option, and configure the
  # subnet for that interface's IPv4 address.
  address: "10.0.0.0/24"
  ...
```

DHCPv6 Links

For DHCPv6, the link is determined by the server according to section 11 of RFC 3315. If the message is received directly and the source address is link-local, then the client is on the link attached to the server interface which received the message. If the message is received directly and the source address is not link-local, then the client is on the link identified by the source address. If the message is received from a relay agent, then the client is on the link identified by the link-address of the Relay-Forward message. Therefore, DHCPv6 link definitions are either local or remote.

Local DHCPv6 Link Example

Local links require the interface element to specify the name of the server interface.

xml

```
<link>
  <name>Local IPv6 Client Link (Multicast traffic)</name>
  <!-- Local DHCPv6 links are defined by interface name -->
  <interface>eth2</interface>
  ...
</link>
```

json

```
{
  "link" : {
    "name" : "Local IPv6 Client Link (Multicast traffic)",
    "interface" : "eth2"
    ...
  }
}
```

yaml

```
link:
  name: "Local IPv6 Client Link (Multicast traffic)"
  # Local DHCPv6 links are defined by interface name
  interface: "eth2"
  ...
```

Remote DHCPv6 Link Example

Remote links require the address element to specify the address of the remote link.

xml

```
<link>
  <name>Remote IPv6 Client Link (Unicast/Multicast traffic)</name>
  <!-- Remote DHCPv6 links are defined in CIDR notation -->
```

```
  <address>2001:db8:2::/48</address>
  ...
</link>
```

json

```
{
  "link" : {
    "name" : "Remote IPv6 Client Link (Unicast/Multicast traffic)",
    "address" : "2001:db8:2::/48"
    ...
  }
}
```

yaml

```
link:
  name: "Remote IPv6 Client Link (Unicast/Multicast traffic)"
  # Remote DHCPv6 links are defined in CIDR notation
  address: "2001:db8:2::/48"
  ...
```

Pools

Pools may be defined only within Links or Link Filters. The Jagornet DHCP Server supports four types of pools.

1. `v4AddrPools` - DHCPv4 address (V4) pools
2. `v6NaAddrPools` - DHCPv6 Non-temporary address (NA) pools
3. `v6TaAddrPools` - DHCPv6 Temporary address (TA) pools
4. `v6PrefixPools` - DHCPv6 Prefix delegation (PD) pools

DHCPv4 Address Pools

DHCPv4 address pools are defined by the `v4AddressPool` type. The address pool must contain a `range` element to define the addresses available for allocation in the pool. The range is specified using the start and end address of a range of addresses. For example:

xml

```
<link>
    <name>DHCPv4 Client Subnet</name>
    <address>10.0.0.0/24</address>
    <v4ConfigOptions>
        <v4SubnetMaskOption>
            <ipAddress>255.255.255.0</ipAddress>
        </v4SubnetMaskOption>
```

```
        <v4RoutersOption>
            <ipAddressList>10.0.0.1</ipAddressList>
            <ipAddressList>10.0.0.2</ipAddressList>
        </v4RoutersOption>
    </v4ConfigOptions>
    <v4AddrPools>
        <poolList>
            <range>10.0.0.100-10.0.0.199</range>
            <configOptions>
                <v4DomainNameOption>
                    <domainName>foo.com.</domainName>
                </v4DomainNameOption>
            </configOptions>
        </poolList>
        <poolList>
            <range>10.0.0.200-10.0.0.254</range>
            <configOptions>
                <v4DomainNameOption>
                    <domainName>bar.com.</domainName>
                </v4DomainNameOption>
            </configOptions>
        </poolList>
    </v4AddrPools>
</link>
```

json

```
{
  "link" : {
    "name" : "DHCPv4 Client Subnet",
    "address" : "10.0.0.0/24",
    "v4ConfigOptions" : {
      "v4SubnetMaskOption" : {
        "ipAddress" : "255.255.255.0"
      },
      "v4RoutersOption" : {
        "ipAddressList" : [ "10.0.0.1", "10.0.0.2" ]
      }
    },
    "v4AddrPools" : {
      "poolList" : [ {
        "range" : "10.0.0.100-10.0.0.199",
        "configOptions" : {
          "v4DomainNameOption" : {
            "domainName" : "foo.com."
          }
        }
      }, {
        "range" : "10.0.0.200-10.0.0.254",
        "configOptions" : {
```

```
          "v4DomainNameOption" : {
            "domainName" : "bar.com."
          }
        }
      } ]
    }
  }
}
```

yaml

```
link:
  name: "DHCPv4 Client Subnet"
  address: "10.0.0.0/24"
  v4ConfigOptions:
    v4SubnetMaskOption:
      ipAddress: "255.255.255.0"
    v4RoutersOption:
      ipAddressList:
      - "10.0.0.1"
      - "10.0.0.2"
  v4AddrPools:
    poolList:
    - range: "10.0.0.100-10.0.0.199"
      configOptions:
        v4DomainNameOption:
          domainName: "foo.com."
    - range: "10.0.0.200-10.0.0.254"
      configOptions:
        v4DomainNameOption:
          domainName: "bar.com."
```

DHCPv6 Address Pools

Non-temporary and temporary address pools are defined by the `addressPool` type. The address pool must contain a `range` element to define the addresses available for allocation in the pool. The range is specified using the start and end address of a range of addresses, or a prefix and length. For example:

xml

```
<link>
    <name>Client Link 2</name>
    <address>2001:DB8:2::/48</address>
    <v6IaNaConfigOptions>
        <v6DnsServersOption>
            <ipAddressList>2001:DB8:2::1</ipAddressList>
        </v6DnsServersOption>
    </v6IaNaConfigOptions>
```

```
    <v6NaAddrPools>
        <poolList>
            <range>2001:DB8:2::0A-2001:DB8:2::FF</range>
            <addrConfigOptions>
                <v6SipServerAddressesOption>
                    <ipAddressList>2001:DB8:2::1:1</ipAddressList>
                </v6SipServerAddressesOption>
            </addrConfigOptions>
        </poolList>
        <poolList>
            <range>2001:DB8:2:1::/64</range>
            <addrConfigOptions>
                <v6SipServerAddressesOption>
                    <ipAddressList>2001:DB8:2:1::1:1</ipAddressList>
                </v6SipServerAddressesOption>
            </addrConfigOptions>
        </poolList>
    </v6NaAddrPools>
</link>
```

json

```
{
  "link" : {
    "name" : "Client Link 2",
    "address" : "2001:DB8:2::/48",
    "v6IaNaConfigOptions" : {
      "v6DnsServersOption" : {
        "ipAddressList" : [ "2001:DB8:2::1" ]
      }
    },
    "v6NaAddrPools" : {
      "poolList" : [ {
        "range" : "2001:DB8:2::0A-2001:DB8:2::FF",
        "addrConfigOptions" : {
          "v6SipServerAddressesOption" : {
            "ipAddressList" : [ "2001:DB8:2::1:1" ]
          }
        }
      }, {
        "range" : "2001:DB8:2:1::/64",
        "addrConfigOptions" : {
          "v6SipServerAddressesOption" : {
            "ipAddressList" : [ "2001:DB8:2:1::1:1" ]
          }
        }
      } ]
    }
  }
}
```

yaml

```
link:
  name: "Client Link 2"
  address: "2001:DB8:2::/48"
  v6IaNaConfigOptions:
    v6DnsServersOption:
      ipAddressList:
      - "2001:DB8:2::1"
  v6NaAddrPools:
    poolList:
    - range: "2001:DB8:2::0A-2001:DB8:2::FF"
      addrConfigOptions:
        v6SipServerAddressesOption:
          ipAddressList:
          - "2001:DB8:2::1:1"
    - range: "2001:DB8:2:1::/64"
      addrConfigOptions:
        v6SipServerAddressesOption:
          ipAddressList:
          - "2001:DB8:2:1::1:1"
```

DHCPv6 Prefix Pools

Prefix delegation pools are defined by the `prefixPool` type. The prefix pool must contain a `range` element to define the prefix available for delegation, and the prefixLength element to define the size of the prefixes to allocate to requesting routers. For example:

xml

```
<link>
    <name>Prefix Delegation Link</name>
    <address>2001:DB8:1::/48</address>
    <v6IaPdConfigOptions>
        <v6DnsServersOption>
            <ipAddressList>2001:DB8:2::1</ipAddressList>
        </v6DnsServersOption>
    </v6IaPdConfigOptions>
    <v6PrefixPools>
        <poolList>
            <range>2001:DB8:2::/48</range>
            <prefixLength>64</prefixLength>
            <prefixConfigOptions>
                <v6SipServerAddressesOption>
                    <ipAddressList>2001:DB8:2:1::1:1</ipAddressList>
                </v6SipServerAddressesOption>
            </prefixConfigOptions>
        </poolList>
    </v6PrefixPools>
```

```
</link>
```

json

```
{
  "link" : {
    "name" : "Prefix Delegation Link",
    "address" : "2001:DB8:1::/48",
    "v6IaPdConfigOptions" : {
      "v6DnsServersOption" : {
        "ipAddressList" : [ "2001:DB8:2::1" ]
      }
    },
    "v6PrefixPools" : {
      "poolList" : [ {
        "range" : "2001:DB8:2::/48",
        "prefixLength" : 64,
        "prefixConfigOptions" : {
          "v6SipServerAddressesOption" : {
            "ipAddressList" : [ "2001:DB8:2:1::1:1" ]
          }
        }
      } ]
    }
  }
}
```

yaml

```
link:
  name: "Prefix Delegation Link"
  address: "2001:DB8:1::/48"
  v6IaPdConfigOptions:
    v6DnsServersOption:
      ipAddressList:
      - "2001:DB8:2::1"
  v6PrefixPools:
    poolList:
    - range: "2001:DB8:2::/48"
      prefixLength: 64
      prefixConfigOptions:
        v6SipServerAddressesOption:
          ipAddressList:
          - "2001:DB8:2:1::1:1"
```

Static Bindings

Static bindings are used to reserve a specific IP address for a specific client. Bindings may be

defined only within Links. The Jagornet DHCP Server supports four types of bindings.

1. v4AddrBindings - DHCPv4 address (V4) bindings
2. v6NaAddrBindings - DHCPv6 Non-temporary address (NA) bindings
3. v6TaAddrBindings - DHCPv6 Temporary address (TA) bindings
4. v6PrefixBindings - DHCPv6 Prefix delegation (PD) bindings

DHCPv4 Address Bindings

DHCPv4 static address bindings are defined by the `v4AddressBinding` type. The address binding must contain a `ipAddress` element to specify the IP address reserved for the client. The address binding must also contain the `chAddr` element to identify the client from the corresponding field in the DHCPv4 request header. For example:

xml

```
<v4AddressBindings>
  <bindingList>
    <!-- Binding addresses should NOT be inside a pool -->
    <ipAddress>10.0.0.200</ipAddress>
    <!-- The MAC address of the client as hex string -->
    <chaddr>0a1b2c3d4e5f</chaddr>
  </bindingList>
</v4AddressBindings>
```

json

```
{
  "v4AddressBindings" : {
    "bindingList" : [ {
      "ipAddress" : "10.0.0.200",
      "chaddr" : "0A1B2C3D4E5F"
    } ]
  }
}
```

yaml

```
v4AddressBindings:
  bindingList:
  # Binding addresses should NOT be inside a pool
  - ipAddress: "10.0.0.200"
    # The MAC address of the client as hex string
    chaddr: "0A1B2C3D4E5F"
```

DHCPv6 Address Bindings

Non-temporary and temporary static address bindings are defined by the `addressBinding` type. The address binding must contain the `ipAddress` element to specify the IP address reserved for the client. The address binding must also contain the `duid` element to identify the client. In addition to the DUID, a static address binding may further identify the client request by specifying the optional `iaid` element. However, if the IA_ID is not specified, then all client requests for the given DUID will be assigned the specified IP address. This may be acceptable if the clients are known to have a single network interface. If the administrator understands the inherent risks with this configuration feature, it may be used with caution. For example:

xml

```
<v6NaAddressBindings>
  <bindingList>
    <ipAddress>2001:db8:1::100</ipAddress>
    <!-- For DHCPv6, clients do not send a MAC address,
         therefore, the DUID can be used for the binding. -->
    <duid>
      <hexValue>0a1b2c3d4e5f</hexValue>
    </duid>
    <!-- The IA_ID may or may not be predictable, so
         it is an optional element for a DHCPv6 binding.  If
         left undefined, then ANY IA_ID will match.
    <iaid>0</iaid>
    -->
  </bindingList>
</v6NaAddressBindings>
```

json

```
{
  "v6NaAddressBindings" : {
    "bindingList" : [ {
      "ipAddress" : "2001:db8:1::100",
      "duid" : {
        "hexValue" : "0A1B2C3D4E5F"
      }
    } ]
  }
}
```

yaml

```
v6NaAddressBindings:
  bindingList:
  - ipAddress: "2001:db8:1::100"
    # For DHCPv6, clients do not send a MAC address,
    # therefore, the DUID can be used for the binding.
```

```
    duid:
      hexValue: "0A1B2C3D4E5F"
```

DHCPv6 Prefix Bindings

Prefix delegation static prefix bindings are defined by the `prefixBinding` type. The prefix binding must contain the `prefix` element to define the prefix available for delegation, and the `prefixLength` element to define the size of the prefixes to allocate to requesting router. The same caveats and cautions as DHCPv6 address bindings apply for prefix bindings as well. For example:

xml

```
<v6PrefixBindings>
  <bindingList>
    <prefix>2001:db8:1::</prefix>
    <prefixLength>64</prefixLength>
    <!-- For DHCPv6, clients do not send a MAC address,
         therefore, the DUID can be used for the binding. -->
    <duid>
      <hexValue>0a1b2c3d4e5f</hexValue>
    </duid>
  </bindingList>
</v6PrefixBindings>
```

json

```
{
  "v6PrefixBindings" : {
    "bindingList" : [ {
      "prefix" : "2001:db8:1::",
      "prefixLength" : 64,
      "duid" : {
        "hexValue" : "0A1B2C3D4E5F"
      }
    } ]
  }
}
```

yaml

```
v6PrefixBindings:
  bindingList:
  - prefix: "2001:db8:1::"
    prefixLength: 64
    duid:
      hexValue: "0A1B2C3D4E5F"
```