Jagornet DHCP Server 4.1

Community Edition User Guide

A. Gregory Rabil

Table of Contents

Introduction . . . 2
- Welcome . . . 2
- Features . . . 2

Getting Started . . . 3
- System requirements . . . 3
- Installation . . . 3
- Running the Server . . . 3
- Logging . . . 9

Configuration . . . 10
- Hierarchy . . . 10
- Server Policies . . . 10
- Channel Resource Policies . . . 11
- Lease Database Policies . . . 12
- Binding Manager Policies . . . 12
- Dynamic DNS Update Policies . . . 13
- DHCP Protocol Policies . . . 18
- DHCPv4 Policies . . . 19
- DHCPv6 Policies . . . 20
- High Availability . . . 21
- REST API . . . 24
- DHCP Options . . . 25
- Generic Options . . . 49
- Filters . . . 61
- Links . . . 79
- Pools . . . 81
- Static Bindings . . . 87

A. Gregory Rabil greg.rabil@jagornet.com Copyright © 2025 Jagornet Technologies, LLC 2025-05-04

Introduction

Welcome

Welcome to Jagornet DHCP Server Community Edition Version 4.1. Please visit these links for the latest news and information on Jagornet DHCP Server.

Jagornet Technologies Website

Jagornet DHCP Community Server on Github

Features

Jagornet DHCP Server is an open source Dynamic Host Configuration Protocol server for IPv4 (DHCPv4) and IPv6 (DHCPv6) implemented in Java. The server supports the following features:

- Standards compliant DHCPv4 server
- Core DHCPv4 RFCs:
  - DHCPv4 - RFC 2131
  - DHCPv4 Options - RFC 2132
- Certified IPv6 Phase II DHCPv6 server - IPv6 Ready Logo
- Core DHCPv6 RFCs:
  - Stateful DHCPv6 - RFC 3315
  - Prefix Delegation - RFC 3633
  - DNS Options - RFC 3646
  - Stateless DHCPv6 - RFC 3736
- Other
  - High Availability (HA) - Primary/Backup failover
  - Dynamic DNS for DHCPv4/v6 - RFC 4703, RFC 4704
  - Highly-scalable, multithreaded architecture using Netty
  - Flexible XML/JSON/YAML configuration file format - Schema
  - Open API 3.0 REST API using Jersey
  - Optimized Docker/OCI Images using Jib

Getting Started

System requirements

Jagornet DHCP Server requires any platform with a certified Java 8+ runtime environment.

The server has been successfully tested on Linux (RHEL/CentOS), Windows and Mac OS using the Oracle JRE 8 and OpenJDK 8 for the respective platforms.

Installation

Java 11+ 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 11+ JRE.

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-dhcpserver-dist-4.1.0-all.tar.gz`
     - `JAGORNET_DHCP_HOME=/opt/jagornet/jagornet-dhcpserver-dist-4.1.0`
   - Windows
     - Installation folder: `C:\jagornet`
     - Compressed install file: `jagornet-dhcpserver-dist-4.1.0-all.zip`
     - `JAGORNET_DHCP_HOME=C:\jagornet\jagornet-dhcpserver-dist-4.1.0`
2. After extracting the compressed files, a jagornet-dhcpserver-dist-4.1.0 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.

Startup Options

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

```
usage: com.jagornet.dhcp.server.JagornetDhcpServer [options]
Jagornet DHCP Server Community Edition 4.1.0
Copyright 2009-2025 Jagornet Technologies, LLC.  All Rights Reserved.
  -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 (default =
                                    file:${jagornet.dhcp.home}/config/dhcpserver
```

```
                                  .xml).
  -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).
  -li,--list-interfaces           Show detailed host interface list, then
                                  exit.
  -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 can be used to operate the server from a command shell. This script supports the following options:

`start [startup options]` - starts the server with any startup options provided.

`stop` - stops the server.

`restart [startup options]` - stop and start the server.

`status` - check if the server is running.

`version` - display server version and exit. A convenience option which can be used instead of 'start -v' or 'start --version'.

`test-configfile <configfile>` - test server configuration file and exit. A convenience option instead of 'start -tc <filename>' or 'start --test-configfile <filename>'.

`list-interfaces` - list host interfaces and exit. A convenience method instead of 'start -li' or 'start --list-interfaces'.

Startup examples:

1. Display the server version and exit (any of the following):

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

```
$JAGORNET_DHCP_HOME/bin/dhcpserver start --version
```

2. Display the startup options and exit (any of the following):

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

3. Start the server with the default options (DHCPv4 unicast on all interfaces, no DHCPv4 broadcast, DHCPv6 unicast on all interfaces, no DHCPv6 multicast):

```
$JAGORNET_DHCP_HOME/bin/dhcpserver start
```

4. 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 start -4p 10067 -4b eth0
```

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

```
$JAGORNET_DHCP_HOME/bin/dhcpserver start -6p 10547 -6m
```

6. 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 start -4b eth0 -6m eth1
```

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

```
$JAGORNET_DHCP_HOME/bin/dhcpserver start -c conf/my-dhcpserver.xml -6u 2001:db8::1
-6m eth0 eth1
```

Windows

Windows Service

Jagornet DHCP Server can operate as a Microsoft Windows Service courtesy of Yet Another Java Service Wrapper (YAJSW). Startup options must be provided in `JAGORNET_DHCP_HOME\bin\yajsw-stable-11.0\conf\wrapper.conf`. Edit this file using a standard text editor (i.e. notepad.exe), and locate the following set of commented properties:

```
# Application parameters.  Add parameters as needed starting from 1
# YAJSW: to specify the main class please use wrapper.java.app.mainclass=
#wrapper.app.parameter.1=
#wrapper.app.parameter.2=
#wrapper.app.parameter.#=
```

Provide desired startup options by adding uncommented `wrapper.app.parameter.#` entries for each option and each option value. For example:

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

```
wrapper.app.parameter.1=-4p
wrapper.app.parameter.2=10067
wrapper.app.parameter.3=-4b
wrapper.app.parameter.4=eth0
```

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

```
wrapper.app.parameter.1=-6p
wrapper.app.parameter.2=10547
wrapper.app.parameter.3=-6m
```

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

```
wrapper.app.parameter.1=-4b
wrapper.app.parameter.2=eth0
wrapper.app.parameter.3=-6m
wrapper.app.parameter.4=eth1
```

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

```
wrapper.app.parameter.1=-c
wrapper.app.parameter.2=config\my-dhcpserver.xml
wrapper.app.parameter.3=-6u
wrapper.app.parameter.4=2001:db8::1
wrapper.app.parameter.5=-6m
wrapper.app.parameter.6=eth0
wrapper.app.parameter.7=eth1
```

The following batch files are provided for operating the Jagornet DHCP Server as a Windows Service.

- `%JAGORNET_DHCP_HOME%\bin\InstallJagornetDhcpServer.bat` - install the Jagornet DHCP Server as a Windows Service. On Windows 7 / Server 2008 this must be "Run As Administrator".
- `%JAGORNET_DHCP_HOME%\bin\UninstallJagornetDhcpServer.bat` - remove the Jagornet DHCP Server as a Windows Service. On Windows 7 / Server 2008 this must be "Run As Administrator".
- `%JAGORNET_DHCP_HOME%\bin\StartJagornetDhcpServer.bat` - start the Jagornet DHCP Server as a Windows Service. The server can also be started using Windows Control Panel → Administrative Tools → System or the Microsoft Management Console (MMC) Services controller.
- `%JAGORNET_DHCP_HOME%\bin\StopJagornetDhcpServer.bat` - stop the Jagornet DHCP Server as a Windows Service. The server can also be stopped using Windows Control Panel → Administrative Tools → System or the Microsoft Management Console (MMC) Services controller.
- `%JAGORNET_DHCP_HOME%\bin\JagornetDhcpServer.bat` - run the server in the command window. Use Ctrl+C to stop.

Command Shell

As an alternative to running Jagornet DHCP Server as a Windows Service, the JAGORNET_DHCP_HOME%\bin\dhcpserver.bat batch file can be used to operate the server from a command shell. Provide any desired startup options on the command line. Enter <Ctrl+C> in the command shell window to stop the server.

Startup examples:

1. Display the server version and exit (any of the following):

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver version
%JAGORNET_DHCP_HOME%\bin\dhcpserver start -v
%JAGORNET_DHCP_HOME%\bin\dhcpserver start --version
```

2. Display the startup options and exit (any of the following):

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver start -?
%JAGORNET_DHCP_HOME%\bin\dhcpserver start --help
```

3. Start the server with the default options (DHCPv4 unicast on all interfaces, no DHCPv4 broadcast, DHCPv6 unicast on all interfaces, no DHCPv6 multicast):

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver start
```

4. 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 start -4p 10067 -4b eth0
```

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

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver start -6p 10547 -6m
```

6. 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 start -4b eth0 -6m eth1
```

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

```
%JAGORNET_DHCP_HOME%\bin\dhcpserver start -c conf\my-dhcpserver.xml -6u 2001:db8::1
-6m eth0 eth1
```

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 for details.

Configuration

Hierarchy

The Jagornet DHCP server is configured via the `config/dhcpserver.xml` XML document. See the Startup Options to change the name and location of the configuration file. The XML schema which defines all server configuration elements is available 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 tables below describes the 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 requests. The following *expert* policies can be adjusted if necessary.

Thread Pool Size

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

Read Buffer Size

- Name: `channel.readBufferSize`
- Default: `307200`

- Description: The size, in bytes, of the network channel read buffer.
- Hierarchy: global

Write Buffer Size

- Name: `channel.writeBufferSize`
- Default: `307200`
- Description: The size, in bytes, of the network channel write buffer.
- Hierarchy: global

Lease Database Policies

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

Schema Type

- Name: `database.schemaType`
- Default: `jdbc-h2`
- Description: The binding database schema type, which can be one of the following:
  - `jdbc-h2` - this default schema type uses JDBC to access an embedded H2 database for lease bindings.
  - `jdbc-derby` - this schema type uses JDBC to access an embedded Apache Derby database for lease bindings.
  - `jdbc-sqlite` - this schema type uses JDBC to access an embedded SQLite database for lease bindings.
- Hierarchy: global

Schema Version

- Name: `database.schemaVersion`
- Default: `2`
- Description: The binding database schema version. Version 1 schema uses the deprecated relational model, and can only be used with the jdbc-* schemaTypes. Version 2 uses a single table model and can be used with all schemaTypes.
- Hierarchy: global

Binding Manager Policies

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

Reaper Startup Delay

- Name: `binding.manager.reaper.startupDelay`
- Default: `10000`
- Description: Number of milliseconds for background thread to wait before checking for expired leases at server startup. Note that bindings are always expired when needed to free them for assignment.
- Hierarchy: global

Reaper Run Period

- Name: `binding.manager.reaper.runPeriod`
- Default: `60000`
- Description: Number of milliseconds for background thread to wait between checks for expired leases. Note that bindings are always expired when needed to free them for assignment.
- Hierarchy: global

Offer Expiration

- Name: `binding.manager.offerExpiration`
- Default: `12000`
- Description: Number of milliseconds after which an offered address is considered free if the address is not requested by the client.
- Hierarchy: global

Delete Old Bindings

- Name: `binding.manager.deleteOldBindings`
- Default: `false`
- Description: Flag to indicate if the server should delete bindings upon expiration, or keep the binding while marking it expired.
- Hierachy: 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 are used to configure the Dynamic DNS update processing.

DDNS Update

- Name: `ddns.update`
- Default: `none`
- Description: Support Dynamic DNS updates for clients which send the Client FQDN option. Available values are:
  - `none` - no DDNS updates
  - `server` - performe all DDNS updates
  - `honorNoUpdate` - honor client FQDN NoUpdate flag
  - `honorNoA` - honor client FQDN NoA flag
  - `honorNoAAAA` - honor client FQDN NoAAAA flag
- Hierarchy: global, filter, link, linkFilter

DDNS Synchronize

- Name: `ddns.synchronize`
- Default: `false`
- Description: Flag to indicate if the server should synchronize DDNS updates with issuing of leases. That is, the DHCP Reply message will not be sent to the client until the DDNS update completes.
- Hierarchy: all

DDNS Domain

- Name: `ddns.domain`
- Description: The domain to use for the client FQDN. If the Client FQDN option in an unqualified hostname, this domain will be appended to the hostname to form the FQDN for DDNS updates. If the Client FQDN contains a domain name, that domain name (everything after the first label, i.e. after the first dot ".") will be replaced by this configured domain name.
- Hierarchy: all

DDNS TTL

- Name: `ddns.ttl`
- Default: `0.3`
- Description: Value for the TTL of DDNS updates. If the value is less than one(1), it is assumed to be a percentage of the valid lifetime in seconds. If the value is greater than or equal to one(1), it assumed to be an absolute number of seconds.
- Hierarchy: all

DDNS Server

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

TSIG Key Name

- Name: `ddns.tsig.keyName`
- Description: The name of the TSIG key for signed DDNS updates.
- Hierarchy: all

TSIG Algorithm

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

TSIG Key Data

- Name: `ddns.tsig.keyData`
- Description: The public key data of the TSIG key in base 64 encoding.
- Hierarchy: all

Forward Zone Name

- Name: `ddns.forward.zone.name`
- Description: The name of the dynamic zone for forward DDNS updates. If not set, the zone will be assumed to be the ddns.domain, or if that is not set, then the portion of the client supplied FQDN which follows the first label.
- Hierarchy: all

Forward Zone TTL

- Name: `ddns.forward.zone.ttl`
- Default: `0.3`
- Description: Value for the TTL of forward DDNS updates. If the value is less than one(1), it is assumed to be a percentage of the valid lifetime in seconds. If the value is greater than or equal to one(1), it assumed to be an absolute number of seconds. This policy is only necessary if the forward DDNS TTL is different from the ddns.ttl policy value.
- Hierarchy: all

Forward Zone Server

- Name: `ddns.forward.zone.server`
- Description: The IP address of the dynamic DNS server for sending forward DDNS updates. This policy is only necessary if the forward DDNS server is different from the ddns.server policy value.
- Hierarchy: all

Forward Zone TSIG Key Name

- Name: `ddns.forward.zone.tsig.keyName`
- Description: The name of the TSIG key for signed forward DDNS updates. This policy is only necessary if the forward DDNS key name is different from the ddns.tsig.keyName policy value.
- Hierarchy: all

Forward Zone TSIG Algorithm

- Name: `ddns.forward.zone.tsig.algorithm`
- Description: The algorithm name used for the TSIG key for signed forward DDNS updates. This policy is only necessary if the forward DDNS algorithm is different from the 'ddns.tsig.algorithm' policy value. Currently supported value is 'hmac-sha256.'
- Hierarchy: all

Forward Zone TSIG Key Data

- Name: `ddns.forward.zone.tsig.keyData`
- Description: The public key data of the TSIG key in base 64 encoding for signed reverse DDNS updates. This policy is only necessary if the forward DDNS key data is different from the ddns.tsig.keyData policy value.
- Hierarchy: all

Reverse Zone Name

- Name: `ddns.reverse.zone.name`
- Description: The name of the dynamic zone for reverse DDNS updates. If not set, the zone will be assumed to be the ip6.arpa domain corresponding to the subnet based on the ddns.reverse.zone.bitLength policy below.
- Hierarchy: all

Reverse Zone Bit Length

- Name: `ddns.reverse.zone.bitLength`
- Default: `64`
- Description: The number of bits representing the subnet for calculating the reverse zone name.

- Hierarchy: all

Reverse Zone TTL

- Name: `ddns.reverse.zone.ttl`
- Default: `0.3`
- Description: Value for the TTL of reverse DDNS updates. If the value is less than one(1), it is assumed to be a percentage of the valid lifetime in seconds. If the value is greater than or equal to one(1), it assumed to be an absolute number of seconds. This policy is only necessary if the reverse DDNS TTL is different from the ddns.ttl policy value.
- Hierarchy: all

Reverse Zone Server

- Name: `dns.reverse.zone.server`
- Description: The IP address of the dynamic DNS server for sending reverse DDNS updates. This policy is only necessary if the reverse DDNS server is different from the ddns.server policy value.
- Hierarchy: all

Reverse Zone TSIG Key Name

- Name: `ddns.reverse.zone.tsig.keyName`
- Description: The name of the TSIG key for signed reverse DDNS updates. This policy is only necessary if the reverse DDNS key name is different from the ddns.tsig.keyName policy value.
- Hierarchy: all

Reverse Zone TSIG Algorithm

- Name: `ddns.reverse.zone.tsig.algorithm`
- Description: The algorithm name used for the TSIG key for signed reverse DDNS updates. This policy is only necessary if the reverse DDNS algorithm is different from the ddns.tsig.algorithm policy value. Currently supported value is 'hmac-sha256.'
- Hierarchy: all

Reverse Zone TSIG Key Data

- Name: `ddns.reverse.zone.tsig.keyData`
- Description: The public key data of the TSIG key in base 64 encoding for signed reverse DDNS updates. This policy is only necessary if the reverse DDNS key data is different from the ddns.tsig.keyData policy value.
- Hierarchy: all

DHCP Protocol Policies

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

Ignore Loopback

- Name: `dhcp.ignoreLoopback`
- Default: `true`
- Description: Ignore the loopback addresses when binding sockets during server startup.
- Hierarchy: global

Ignore Link Local

- Name: `dhcp.ignoreLinkLocal`
- Default: `true`
- Description: Ignore the link local addresses when binding sockets during server startup.
- Hierarchy: global

Ignore Self Packets

- Name: `dhcp.ignoreSelfPackets`
- Default: `true`
- Description: Ignore packets received from one of the server's addresses.
- Hierarchy: global

Recent Message Timer

- Name: `dhcp.processor.recentMessageTimer`
- Default: `5000`
- Description: Number of milliseconds to keep track of recent messages. Used to minimize replays of the same message to the server. That is, to help mitigate denial of service (DOS) attacks.
- Hierarchy: global

Send Requested Options Only

- Name: `dhcp.sendRequestedOptionsOnly`
- Default: `false`
- Description: Flag to indicate if the server should return only the options requested by a client in the Option Request Option (ORO) if available, or send all configured options.
- Hierarchy: all

Support Rapid Commit

- Name: `dhcp.supportRapidCommit`
- Default: `false`
- Description: Flag to indicate if the server should support clients requesting rapid commit of binding.
- Hierarchy: global, filter, link, linkFilter

DHCPv4 Policies

Header: Server Name

- Name: `v4.header.sname`
- Description: The server host name field of the DHCPv4 header. Used in conjunction with v4.header.filename. See also - v4TftpServerNameOption.
- Hierarchy: all

Header: Bootfile Name

- Name: `v4.header.filename`
- Description: The boot file name field of the DHCPv4 header. The name of a boot file which the client will retrieve from the server specified in the sname header field. See also - v4BootFileNameOption.
- Hierarchy: all

Ignored MAC Addresses

- Name: `v4.ignoredMacAddrs`
- Default: `000000000000, FFFFFFFFFFFF`
- Description: A list of comma separated MAC addresses for the server to ignore requests from.
- Hierarchy: all

Default Lease Time

- Name: `v4.defaultLeasetime`
- Default: `3600`
- Description: The lease time for DHCPv4 clients.
- Hierarchy: all

Ping Check Timeout

- Name: `v4.pingCheckTimeout`
- Default: `0`

- Description: The number of milliseconds to wait for a response to a ping before offering new addresses to DHCPv4 clients.
- Hierarchy: global

DHCPv6 Policies

Preferred Lifetime

- Name: `v6.preferredLifetime`
- Default: `3600`
- Description: Number of seconds for the preferred lifetime of addresses/prefixes provided by the server to a DHCPv6 client.
- Hierarchy: global, link, pool

Valid Lifetime

- Name: `v6.validLifetime`
- Default: `3600`
- Description: Number of seconds for the valid lifetime of addresses/prefixes provided by the server to a DHCPv6 client.
- Hierarchy: global, link, pool

Verify Uknown Rebind

- Name: `v6.verifyUnknownRebind`
- Default: `false`
- Description: Flag to indicate if the server should attempt to verify that addresses in a DHCPv6 client's request are appropriate for the client's link, even though that client is unknown to the server. See section 18.2.4 of RFC 3315.
- Hierarchy: global, filter, link, linkFilter

IA NA T1

- Name: `v6.iaNaT1`
- Default: `0.5`
- Description: Percentage of shortest preferred lifetime of DHCPv6 addresses in the IA_NA to set the IA_NA T1 (renew) time in server replies.
- Hierarchy: global, link

IA NA T2

- Name: `v6.iaNaT2`

- Default: `0.8`
- Description: Percentage of shortest preferred lifetime of DHCPv6 addresses in the IA_NA to set the IA_NA T2 (rebind) time in server replies.
- Hierarchy: global, link

IA PD T1

- Name: `v6.iaPdT1`
- Default: `0.5`
- Description: Percentage of shortest preferred lifetime of DHCPv6 prefixes in the IA_PD to set the IA_PD T1 (renew) time in server replies.
- Hierarchy: global, link

IA PD T2

- Name: `v6.iaPdT2`
- Default: `0.8`
- Description: Percentage of shortest preferred lifetime of DHCPv6 addresses in the IA_PD to set the IA_PD T2 (rebind) time in server replies.
- Hierarchy: global, link

High Availability

Jagornet DHCP Server supports High Availability (HA). The implementation is a simple "warm-standby" backup mechanism. This is not the same as other failover implementations. Instead, HA is attained through a process which involves the following:

- DHCP Relays configured with IP address of both Primary and Backup Jagornet DHCP servers
- Primary and Backup Jagornet DHCP servers have identical configurations except for the HA related policies described below
- Primary is started, gives out leases
- Backup comes online, syncs all leases from Primary
- Backup starts polling loop to check Primary operational status
- Primary handles all lease requests
- Backup ignores all lease requests while poll requests are answered by Primary
- If poll failures reach threshold defined by HA policies below, then Backup becomes active and starts handling lease requests
- Primary comes back online, syncs lease changes from backup
- Primary takes over lease handling as each link is sync'd
- In the event that the Primary failure was catastrophic, or in situations where the lease database has been lost or has been corrupt, then the Primary can be forced to sync all leases from the

backup, instead of just those leases that are new or changed since the Primary went offline. To do so, simply delete the file defined for the `ha.stateDbFile` policy below before starting the Primary.

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

HA Role

- Name: `ha.role`
- Description: High Availability (HA) Role:
  - `primary` - active server
  - `backup` - standby server
- Hierarchy: global

HA Protocol

- Name: `ha.protocol`
- Default: `grpc`
- Description: High Availability (HA) Protocol:
  - `grpc` - uses Mutual TLS (mTLS) for authentication
  - `rest` - uses username/password for authentication, see policies below
- Hierarchy: global

HA Binding Update Mode

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

HA Peer Server

- Name: `ha.peerServer`
- Description: The IP address of the HA peer server
- Hierarchy: global

HA Peer Port

- Name: `ha.peerPort`

- Default: `9066`
- Description: The port of the HA peer server
- Hierarchy: global

HA Poll Seconds

- Name: `ha.pollSeconds`
- Default: `30`
- Description: The number of seconds between poll messages to HA peer server
- Hierarchy: global

HA Poll Reply Timeout

- Name: `ha.pollReplyTimeout`
- Default: `1000`
- Description: The number of milliseconds to wait for a poll reply from HA peer server
- Hierarchy: global

HA Poll Reply Failure Count

- Name: `ha.pollReplyFailureCount`
- Default: `5`
- Description: The number of poll reply failures before considering the HA peer server unavailable
- Hierarchy: global

HA Request All Leases On Restart

- Name: `ha.requestAllLeasesOnRestart`
- Default: `true`
- Description: Flag to request all leases on restart, or only those that have changed since last communication with HA peer server
- Hierarchy: global

HA Username

- Name: `ha.username`
- Default: `hapeer`
- Description: High Availability (HA) username. Ensure that the `ha.peerUsername` configured on the peer server matches this value. *For REST HA protocol only.*
- Hierarchy: global

HA Password

- Name: `ha.password`
- Default: `jagornet`
- Description: High Availability (HA) password. Ensure that the `ha.peerPassword` configured on the peer server matches this value. *For REST HA protocol only.*
- Hierarchy: global

HA Peer Username

- Name: `ha.peerUsername`
- Default: `hapeer`
- Description: High Availability (HA) peer username. Ensure that this value matches the `ha.username` configured on the peer server. *For REST HA protocol only.*
- Hierarchy: global

HA Peer Password

- Name: `ha.peerPassword`
- Default: `jagornet`
- Description: High Availability (HA) peer password Ensure that this value matches the `ha.password` configured on the peer server. *For REST HA protocol only.*
- Hierarchy: global

HA State DB File

- Name: `ha.stateDbFile`
- Default: `db/ha/jagornet-ha-state.db`
- Description: The HA state database filename
- Hierarchy: global

HA Max Stored States

- Name: `ha.maxStoredStates`
- Default: `10`
- Description: The number of previous states maintained in the 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://{jagornet-

dhcp-server-name-or-ip}:9067. Please view the Open API 3.0 documentation in the Swagger Editor:

WARNING	The dhcpleases POST/PUT/DELETE endpoints are intended for internal use only!

View Jagornet DHCP Server Open API docs in Swagger Editor

REST API Username

- Name: `rest.api.username`
- Default: `jagornet`
- Description: The REST API username
- Hierarchy: global

REST API Password

- Name: `rest.api.password`
- Default: `jagornet`
- Description: The REST API password
- Hierarchy: global

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 Community Edition 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>s
        </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"
```