

# Dibbler – a portable DHCPv6 User's guide

Tomasz Mrugalski  
[thomson\(at\)klub.com.pl](mailto:thomson(at)klub.com.pl)

2004-12-27

0.3.1

## Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Intro</b>                            | <b>3</b>  |
| <b>2</b> | <b>Overview</b>                         | <b>3</b>  |
| <b>3</b> | <b>Installation and usage</b>           | <b>4</b>  |
| 3.1      | Linux installation . . . . .            | 4         |
| 3.2      | Windows installation . . . . .          | 5         |
| <b>4</b> | <b>Compilation</b>                      | <b>5</b>  |
| 4.1      | Linux compilation . . . . .             | 5         |
| 4.2      | WindowsXP compilation . . . . .         | 6         |
| <b>5</b> | <b>Config files</b>                     | <b>6</b>  |
| 5.1      | Tokens and basic informations . . . . . | 6         |
| 5.2      | Scopes . . . . .                        | 7         |
| 5.3      | Comments . . . . .                      | 7         |
| 5.4      | Client config file . . . . .            | 7         |
| 5.4.1    | Global scope . . . . .                  | 7         |
| 5.4.2    | Interface declaration . . . . .         | 8         |
| 5.4.3    | IA declaration . . . . .                | 8         |
| 5.4.4    | Address declaration . . . . .           | 8         |
| 5.4.5    | Standard options . . . . .              | 9         |
| 5.4.6    | Additional options . . . . .            | 9         |
| 5.4.7    | Stateless configuration . . . . .       | 10        |
| 5.4.8    | Example client config file . . . . .    | 10        |
| 5.5      | Server config file . . . . .            | 12        |
| 5.5.1    | Global scope . . . . .                  | 12        |
| 5.5.2    | Interface declaration . . . . .         | 12        |
| 5.5.3    | Class scope . . . . .                   | 13        |
| 5.5.4    | Options . . . . .                       | 13        |
| 5.5.5    | Additional options . . . . .            | 14        |
| 5.5.6    | Example server config file . . . . .    | 14        |
| <b>6</b> | <b>Frequently Asked Question</b>        | <b>16</b> |
| 6.1      | Common . . . . .                        | 16        |
| 6.2      | Linux specific . . . . .                | 17        |
| 6.3      | Windows specific . . . . .              | 17        |
| <b>7</b> | <b>History</b>                          | <b>17</b> |
| <b>8</b> | <b>Contact</b>                          | <b>17</b> |
| <b>9</b> | <b>Thanks and greetings</b>             | <b>18</b> |

## 1 Intro

First of all, as an author I would like to thank you for your interest in this DHCPv6 implementation. If this documentation doesn't answer your question or you have any suggestions, feel free to contact me. See *Contact* section for details. Also be sure to check out Dibbler website located at <http://klub.com.pl/dhcpv6/>.

## 2 Overview

Dibbler is a portable DHCPv6 solution. It features both server and client. Currently there are ports available for Windows XP and 2003 and Linux systems. Besides basic functionality specified in RFC3315, it also offers several enhancements, e.g. DNS servers and domain names configuration.

Dibbler is an open source software, distributed under GNU GPL licence. It means that it is freely available, free of charge and can be used by anyone (including commercial users). Sources are also available, so anyone skilled enough can fix bugs or add new features.

As for now, Dibbler offers these features:

- Basic server discovery and address assignment (SOLICIT, ADVERTISE, REQUEST and REPLY messages) – simplest case possible: client discovers server, then asks for an address, which is granted by a server.
- Best server discovery – when client receives more than one ADVERTISE messages from different servers, it chooses the best one and remembers remaining ones as a backup.
- Many servers support – client is capable of discovering and dealing with multiple servers. For example, client would like to have 5 addresses configured. Preferred server can only grant 3, so client send request for remaining 2 addresses to one of the remaining servers.
- Unicast communication – if specific conditions are met, client could send messages directly to a server's unicast address, so additional servers does not need to process those messages. It also improves efficiency, as all nodes present in LAN segment receive multicast packets.<sup>1</sup>
- Address renewal (RENEW,REBIND and REPLY messages) – client renews addresses at certain time intervals, if server specified so.
- Duplicate address detection (DECLINE and REPLY messages) – client can detect and properly handle faulty situation, when server grants address which is illegally used by some other host. It will inform server of such circumstances, and request for another address. Server will mark this address as used by unknown host, and will assign another address to a client.
- Power failure/crash support (CONFIRM and REPLY messages) – after client recovers from crash or power failure, it still can have assigned valid addresses. In such circumstances, client uses CONFIRM message, to config if those addresses are still valid<sup>2</sup>.
- IA Option – this option is used to carry addresses. Both server and client support multiple IAs in one message. Additional feature is client capability to ask for a specific address.
- Rapid Commit Option (SOLICIT and REPLY messages) – if both client and server are configured to use rapid commit, address assignment procedure can be shortened to 2 messages. Major advantage is lesser network usage and quicker client startup time.

---

<sup>1</sup>Nodes, which do not belong to specific multicast group, drop those packets silently. However, determining if host belongs or not to a group must be performed on each node.

<sup>2</sup>As for 0.2.1 version, this functionality works on server side only, client side support will be available in 0.2.2.

Except RFC3315-specified behavior, Dibbler also support several enhancements:

- DNS Servers Option – client can ask for information about DNS servers.
- Domain Name Option – client can ask for information about domain name it is connected in.
- Time Zone Option – client can ask for information about time zone it is currently in.
- NTP Servers Option – client can ask for Network Time Protocol Servers to synchronize its clock.

And now some implementation specific details:

- On both server and client, after each modification data is dumped in XML format, so it can be easily processed in an automated manner. Simple example of this advantage is a script, which can generate reports about server usage (assigned addresses, clients configured and so on);
- Dibbler is fully portable. Core logic is system independent and coded in C++ language. There is about dozen low-level functions, which are system specific. They're used for adding addresses, retrieving information about interfaces, setting DNS servers and so on. Porting Dibbler to other systems (and even other architectures) would require implement only those several system-specific functions.

See RELEASE-NOTES for details about version-specific upgrades, fixes and features.

## 3 Installation and usage

Both client and server are installed in the same way. Installation method is different in WindowsXP and Linux systems, so they're described separately. To simplify installation, it assumes that binary versions are used<sup>3</sup>.

### 3.1 Linux installation

Obtain (e.g. download from <http://klub.com.pl/dhcpv6/>) an archive with Dibbler binaries and extract it to `/var/lib/dibbler` directory:

```
cd /var/lib/dibbler/  
tar zxvf dibbler-0.2.1-linux.tar.gz
```

Depending what functionality do you want to use (server or client), you should edit config file (`client.conf` for client and `server.conf` for server). After editing, issue following commands:

```
./dibbler-server start
```

for starting server or

```
./dibbler-client start
```

for starting client. `start` parameter needs a little comment. It instructs Dibbler to run in daemon mode – detach from console and run in the background. During config files fine-tuning, it is often better to watch Dibbler's behavior instantly. In this case, use `run` instead of `start` parameter. Dibbler will present its messages on your console. To finish it, press `ctrl-c`.

To stop server running in daemon mode, type:

---

<sup>3</sup>Compilation is not required, binary version can be used safely. Compilation can be performed by advanced users, see *Compilation* section for details.

```
./dibbler-server stop
```

To stop client running in daemon mode, type:

```
./dibbler-client stop
```

To see, if client or server are running<sup>4</sup>, type:

```
./dibbler-client status
```

or

```
./dibbler-server status
```

## 3.2 Windows installation

Starting at 0.2.1, Dibbler supports Windows XP and 2003. Obtain (e.g. download from <http://klub.com.pl/>) an archive with Dibbler binaries and extract it to a directory, e.g. `c:\dibbler`.

Open a console (start → run... → cmd) and issue following commands:

```
c:
cd \dibbler
dibbler-client run -d c:\dibbler
```

`run` command instructs Dibbler to run in a console and not become a daemon. Is it very useful feature in early configuration stages. To finish it, press ctrl-c.

After configuration files are edited and tested, user can install Dibbler as a Windows service. In this case, following command should be issued:

```
dibbler-client install -d c:\dibbler
```

Now dibbler is installed as a Windows service and it can be controlled from Control Panel → Administrative tasks → Services.

If you want to uninstall Dibbler as a service, use `uninstall` instead of `install`.

## 4 Compilation

Dibbler is distributed in 2 versions: binary and source files. For most users, binary version is better choice. Compilation is performed by more experienced users, preferably with programming knowledge. It does not offer any advances over binary version, only allows to understand internal Dibbler workings. You probably want just install and use Dibbler. If that is your case, read section named *Installation*.

### 4.1 Linux compilation

Compilation in most cases is not necessary and should be performed only by experienced users. Preferred method is to use binaries provided on Dibbler's website. Issue following commands:

```
tar zxvf dibbler-0.3.1-src.tar.gz
cd dibber
make server client
```

That's it. You can also install it in the system by issuing command:

---

<sup>4</sup>Running status is based on `/var/lib/dibbler/client.pid` or `server.pid` files. In rare occasions, when server crashes, this status will show server status as running.

`make install`

If there are problems with missing/different compiler version, take a look at the beginning of the `Makefile.inc` file. Dibbler was compiled using gcc 2.95, 3.0, 3.2, 3.3 and 3.4 versions. Lexer files were generated using flex 2.5.31. Parser file were created using bison++ 1.21.9<sup>5</sup>. Everything was developed under Debian GNU/Linux system.

If there are problems with `SrvLexer.cpp` and `ClntLexer.cpp` files, please use `FlexLexer.h` in `port-linux/` directory. Most simple way to do this is to copy this file to `/usr/include` directory. Additional information about compilation can be found in *Dibbler Developer's Guide*.

## 4.2 WindowsXP compilation

Download `dibbler-0.3.0-src.tar.gz` and extract it. In `port-winxp` there will be project files (one for server and one for client) for MS Visual C++ 2003. Open one of them and click Build command. That should do the trick. Additional information about compilation can be found in *Dibbler Developer's Guide*.

## 5 Config files

This section describes Dibbler server and (optional) client configuration. Square brackets denotes optional values: mandatory [optional]. Alternative is marked as |. `A | B` means A or B.

Parsers are case-insensitive, so `Iface`, `IfAcE`, `iface` and `IFACE` mean the same. This does not apply to interface names, of course. `eth0` and `ETH0` are two different interfaces.

### 5.1 Tokens and basic informations

Config file parsing is token-based. Here's list of tokens used:

**IPv6 address** – IPv6 address

**32-bit decimal integer** – string containing only numbers, e.g. 123456

**string** – string of arbitrary characters enclosed in single or double quotes, e.g. 'this is string'. If string contains only a-z, A-Z and 0-9 characters, quotes can be omitted.

**DUID identifier** – hex number starting with 0x, e.g. 0x12abcd.

**IPv6 address list** – IPv6 addresses separated with commas.

**DUID list** – DUIDs separated with commas.

**string list** – strings separated with commas.

**Boolean** – YES, NO, TRUE, FALSE, 0 or 1. Each of them can be used, when user must enable or disable specific option.

There are also some client or server specified tokens. Here's the list:

**client** – `iface`, `address`, `ia`, `no-ia`, `log-mode`, `log-name`, `log-level`, `work-dir`, `preferred-lifetime`, `valid-lifetime`, `t1`, `t2`, `reject-servers`, `preferred-servers`, `rapid-commit`, `unicast`, `sip-server`, `sip-domain`, `nis-server`, `nis-domain`, `nis+-server`, `nis+-domain`, `lifetime`, `dns-server`, `domain`, `ntp-server`, `time-zone`,

**server** – `iface`, `class`, `log-level`, `log-name`, `log-level`, `work-dir`, `accept-only`, `reject-only`, `t1`, `t2`, `preferred-lifetime`, `valid-lifetime`, `unicast`, `preference`, `pool`, `rapid-commit`, `client-max-lease`, `class-max-lease`, `iface-max-lease`, `dns-servers`, `domain`, `ntp-server`, `time-zone`, `sip-server`, `sip-domain`, `nis-server`, `nis-domain`, `nis+-server`, `nis+-domain`, `lifetime`

---

<sup>5</sup>flex and bison++ tools are not required to compile Dibbler. Generated files are placed in CVS and in tar.gz archives

## 5.2 Scopes

There are four scopes, in which options can be specified:

- global
- interface
- IA
- address

Global scope is the largest. It covers the whole config file and applies to all interfaces, IAs, and addresses, until some lower scope options override it. Next comes interface scope. Options defined there are interface-specific and apply to this interface, all IAs in this interface and addresses in those IAs. Next is IA scope. Options defined there are IA-specific and apply to this IA and to addresses it contains. Least significant scope is address. Every option is specific for one scope. For example, T1 is defined for IA scope. However, it can be also used in more common scopes. In this case – in interface or global. Defining T1 in interface scope means: „for this interface default value for T1 is ...”. The same applies to global scope. Options can be used multiple times. In that case value defined later is used.

## 5.3 Comments

Comments are also allowed. All common comment styles are supported:

- C++ style one-line comments: `// this is comment`
- C style multi-line comments: `/* this is multiline comment */`
- bash style one-line comments: `# this is one-line comment`

## 5.4 Client config file

Client config file should be named `client.conf`. After successful startup, old version of this file is stored as `client.conf-old`. One of design requirements for client was „out of the box” usage. To achieve this, simply use empty `client.conf` file. Client will try to get one address for each up and running interface <sup>6</sup>.

### 5.4.1 Global scope

Every option can be declared in global scope. Config file has this form:

```
interface declaration |
global options        |
interface options     |
IA options            |
address options
```

---

<sup>6</sup>Exactly: Client tries to configure each up, multicast-capable and running interface, which has link address at least 6 bytes long. So it will not configure tunnels (which usually have IPv4 address (4bytes long) as their link address. It should configure all Ethernet and 802.11 interfaces. The latter was not tested by author due to lack of access to 802.11 equipment.

### 5.4.2 Interface declaration

Interface can be declared this way:

```
iface name_of_this_interface
{
    interface options      |
    IA options             |
    address options
}
```

or

```
iface number
{
    interface options      |
    IA options             |
    address options
}
```

In every case, number denotes interface number. It can be extracted from „ip l” (Linux) or „ipv6 if” (Windows). `name_of_this_interface` is an interface name. Also take a note that name of the interface no longer needs to be enclosed in single or double quotes. It is necessary only in Windows systems, where interface names sometimes contain spaces, e.g. "local network connection".

### 5.4.3 IA declaration

IA is declared this way:

```
ia [number]
[
    [address declaration  |
    IA options            |
    address options       ]
    ...
]
```

where number is an optional number, which describes how many such IAs should be requested. If this number is not equal 1, then address options are not allowed. That could come in handy when someone need several IAs with the same parameters. If IA contains no addresses, client assumes that one address should be configured.

### 5.4.4 Address declaration

Addres is declared like this:

```
address [number]
[
    [address options      |
    IPv6 address          ]
]
```

where number denotes how many addresses with those values should be requested. If it is diffrent than 1, then IPv6 address options are not allowed.



### 5.4.5 Standard options

Standard options are... well, standard. This means that they have nothing to do with any extensions. Standard options are declared this way:

```
OptionName option-value
```

Every option has a scope it can be used in, default value and sometimes allowed range. Parameters denoted with (H) are used as hints for the server. Value of *work-dir* option is currently not used. In *log-mode* option, *short* and *full* values are supported. *syslog* and *eventlog* will be available in future releases. *rapid-commit* and *unicast* expect one boolean parameter. It can be TRUE, FALSE, YES, NO, 0 or 1. Setting log level to too low value (5 or less) can result in mysterious behavior. 6 or 7 is a recommended value.

| Name               | Scope     | Values (default)   | default    | Description  |
|--------------------|-----------|--------------------|------------|--|
| valid-lifetime     | address   | integer            | 4294967296 | valid lifetime for address (specified in seconds) (H)  |
| preferred-lifetime | address   | integer            | 4294967296 | after this amount of time(in seconds) address becomes depreciated (H)                            |
| T1                 | IA        | integer            | 4294967296 | client should renew addresses after T1 seconds (H)   |
| T2                 | IA        | integer            | 4294967296 | client should send REBIND after T2 seconds (H)   |
| reject-servers     | IA        | addrs or DUID list | empty      | list containing servers which should be discarded in configuration of this IA                    |
| preferred-servers  | IA        | addrs or DUID list | empty      | Preferred servers list. ADVERTISE messages received by client are sorted according to this list. |
| rapid-commit       | interface | 0 or 1             | 0          | should we use Rapid Commit?  |
| unicast            | interface | 0 or 1             | 0          | Is unicast communication allowed?  |
| work-dir           | global    | string             | empty      | working directory  |
| log-level          | global    | 1-8                | 8          | log-level (8 is most verbose)  |
| log-name           | global    | string             | Client     | Name, which appears in a log file  |
| log-mode           | global    | short or full      | full       | logging mode: short (date and name suppressed) or full.  |

### 5.4.6 Additional options

Additional options are the options specified in external drafts and in RFC documents. They are declared with *option* keyword:

```
option OptionName option-value
```

where OptionName is one of possible values listed below:

| OptionName  | Scope     | Values (default) | default | Description                            |
|-------------|-----------|------------------|---------|--|
| dns-server  | interface | addrs list       | empty   | preferred DNS servers list (H)         |
| domain      | interface | domains list     | empty   | preferred domain (H)                   |
| ntp-server  | interface | addrs list       | empty   | preferred NTP servers list (H)         |
| time-zone   | interface | timezone         | empty   | preferred time zone (H)                |
| sip-server  | interface | addrs list       | empty   | preferred SIP servers list (H)         |
| sip-domain  | interface | domains list     | empty   | preferred SIP domain (H)               |
| nis-server  | interface | addrs list       | empty   | preferred NIS servers list (H)         |
| nis-domain  | interface | domain           | empty   | preferred NIS domain (H)               |
| nis+-server | interface | addrs list       | empty   | preferred NIS+ servers list (H)        |
| nis+-domain | interface | domain           | empty   | preferred NIS+ domain (H)              |
| lifetime    | interface | YES/NO           | no      | Should client request lifetime option? |

Note that timezone format is described in file `draft-ietf-dhc-dhcpv6-opt-tz-00.txt` and domain format is described in RFC 3646. After receiving options values from a server, client stores them in separate files in the working directory, e.g. `option-dns-server`. Several options are processed and set up in the system. Options supported in Linux and Windows environments are presented in the table below.

| Option      | Support (Linux) | Support (winXP/2003) |
|-------------|-----------------|----------------------|
| dns-server  | system, file    | system, file         |
| domain      | file            | system, file         |
| ntp-server  | file            | file                 |
| time-zone   | file            | file                 |
| sip-server  | file            | file                 |
| sip-domain  | file            | file                 |
| nis-server  | file            | file                 |
| nis-domain  | file            | file                 |
| nis+-server | file            | file                 |
| nis+-domain | file            | file                 |

#### 5.4.7 Stateless configuration

If interface does not contain IA keyword, one IA with one address is assumed. If client should not request for address on this interface, *stateless*<sup>7</sup> must be used. In such circumstances, only specified options will be requested.

#### 5.4.8 Example client config file

In simplest case, client config can be empty. Client will try to assign one address for every interface present in the system, except interfaces:

- which are down (flag UP not set)
- loopback (flag LOOPBACK set)
- which are not running (flag RUNNING not set)

<sup>7</sup>In the version 0.2.1-RC1 and earlier, this directive was called no-ia. This depreciated name is valid for now, but might be removed in future releases.

- which are not multicast capable (flag MULTICAST not set)

If you must use DHCPv6 on one of such interfaces (which is not recommended and probably will fail), you must explicitly specify this interface in config file.

Simple config file requesting 1 address and DNS configuration on eth0 interface looks like that:

```
log-mode short
log-level 7
iface eth0 {
    option dns-server
    ia {
    }
}
```

Another example is presented below. Client asks for 1 address and would like it to be 2000::1:2:3. Rapid-commit is allowed and client would like to renew this address once in a 10 minutes:

```
log-mode short
log-level 7
iface eth0 {
    rapid-commit YES
    ia {
        address {
            2000::1:2:3
        }
    }
}
```

Here's yet another example. We would like to obtain 2 addresses on „Local Area Connection” interface. Unicast communication is ok in this scenario. We don't care for details, so keep those log very short. Config file looks like that:

```
log-mode short
log-level 3
iface "Local Area Connection" {
    ia {
        address
        address
    }
}
```

Here is a more complicated case. Let's say there are 4 interfaces, numbered 1 thru 4. Interfaces 1,2 and 3 are not to be configured. Interface 4, named eth0 should have 3 IAs. Two of them are supposed to contain one address each. Third IA should contain 3 addresses. Addresses assigned to first and second IA should have preferred-lifetime 1 hour and valid-lifetime 2 hours. This IA should have 3 specific addresses: 2000::1, 2000::2 and 2000::3. Information about NTP servers,our current timezone, available DNS servers and our domain should also be retrieved. Here's config file:

```
iface eth0
{
    valid-lifetime 7200    // default value - 2 hours
    preferred-lifetime 3600 // default value - 1 hour
```

```
T1 600                // request 10 minutes interval
T2 1200 /* trouble begins after 20 minutes
           of server's silence */
IA 2                  // 2 IAs with just specified values (1h/2h/10min/20min)
IA // third IA is more specific
{
    valid-lifetime 3600 // valid lifetime changed to 1 hour
    preferred-lifetime 1800 // preferred lifetime changed to 30min
    address
    {
        2000::1 // request those addresses
        2000::2
        2000::3
    }
}
option ntp-servers      // ask for NTP servers
option time-zone        // ask for timezone
option dns-servers      // ask for DNS servers
option domain           // ask for domain
}
```

## 5.5 Server config file

Server configuration is stored in `server.conf` file. After successful startup, old version of this file is stored as `server.conf-old`.

### 5.5.1 Global scope

Every option can be declared in global scope. Config file has this form:

```
interface declaration |
global options        |
interface options      |
class options
```

### 5.5.2 Interface declaration

Interface can be declared this way:

```
iface name_of_this_interface
{
    interface options      |
    class options
}
```

or

```
iface number
{
    interface options      |
    class options
}
```

where `name_of_this_interface` denotes name of the interface and number denotes it's number. It no longer needs to be enclosed in single or double quotes (except windows cases, when interface name contains spaces).

### 5.5.3 Class scope

Address class is declared as follows:

```
class
{
    class options |
    address poll
}
```

address poll has this format:

```
poll minaddress-maxaddress
```

### 5.5.4 Options

Every option has a scope it can be used in, default value and sometimes allowed range.

| Name               | Scope     | Values (default)   | default    | Description   |
|--------------------|-----------|--------------------|------------|---|
| work-dir           | global    | string             | empty      | working directory   |
| log-level          | global    | 1-8                | 8          | log-level (8 is most verbose)   |
| log-name           | global    | string             | Client     | Name, which appears in a log file   |
| log-mode           | global    | short or full      | full       | logging mode: short (date and name suppressed) or full                        |
| preference         | interface | 0-255              | 0          | server preference value (higher is more preferred)                            |
| unicast            | interface | address            | empty      | Specify which address should be used.   |
| iface-max-lease    | interface | integer            | 4294967296 | how many addresses can be leased by all clients?                              |
| client-max-lease   | interface | integer            | 4294967296 | how many addresses can be leased by one client?                               |
| rapid-commit       | interface | 0 or 1             | 0          | should we allow Rapid Commit (SOLICIT-REPLY)?                                 |
| valid-lifetime     | class     | integer            | 4294967296 | valid lifetime for address (specified in seconds)                             |
| preferred-lifetime | class     | integer            | 4294967296 | after this amount of time(in seconds) address becomes depreciated             |
| T1                 | class     | integer            | 4294967296 | client should renew addresses after T1 seconds                                |
| T2                 | class     | integer            | 4294967296 | client should send REBIND after T2 seconds                                    |
| reject-clients     | class     | addrs or DUID list | empty      | list containing servers which should be discarded in configuration of this IA |
| accept-only        | class     | addrs or DUID list | empty      | these are the only clients allowed to use this class                          |
| class-max-lease    | class     | integer            | 4294967296 | how many addresses can be leased from this class?                             |

### 5.5.5 Additional options

Server supports additional options, not specified in RFC3315. They have generic form:

option OptionName OptionsValue

All supported options are specified in the table below:

| OptionName     | OptionsValue | Default | Description              |
|----------------|--------------|---------|--------------------------|
| dns-server     | addrs list   | empty   | DNS servers list         |
| domain         | string list  | empty   | domain names list        |
| ntp-server     | addrs list   | empty   | NTP servers list         |
| time-zone      | timezone     | empty   | time zone                |
| sip-server     | addrs list   | empty   | SIP servers list         |
| sip-domain     | string list  | empty   | domain names list        |
| nis-server     | addrs list   | empty   | NIS servers list         |
| nis-domain     | string       | empty   | domain name              |
| nisplus-server | addrs list   | empty   | NIS+ servers list        |
| nis-domain     | string       | empty   | domain name              |
| lifetime       | integer      | empty   | how often renew options? |

Lifetime is a special case. It is not set up by client in a system configuration. It is, however, used by the client to know how long obtained values are correct.

### 5.5.6 Example server config file

In opposite to client, server uses only interfaces described in config file. Let's examine this common situation: server has interface named *eth0* (which is fourth interface in the system) and is supposed to assign addresses from 2000::100/124 class. Simplest config file looks like that:

```
iface eth0
{
    class
    {
        pool 2000::100-2000::10f
    }
}
```

Another example: Server should support 2000::0/120 class on eth0 interface. It should not allow any client to obtain more than 5 addresses and should not grant more than 50 addresses in total. And it should have preference set to 7, accept T1 from 1 to 2 minutes and so on. Config file is presented below:

```
log-mode short
log-level 7
iface eth0
{
    iface-max-lease 50
    client-max-lease 5
    preference 7
    class
    {
        pool 2000::1-2000::100
    }
}
```

Here's modified previous example. Instead of specified limits, unicast communication should be supported.

```
log-level 7
iface eth0
{
    unicast 2000::1234
    class
    {
        pool 2000::1-2000::100
    }
}
```

Ok, now let's present last config file with various requirements:

- Add new class on fifth interface named *eth1*, for example 2000::fe00/120.
- Rapid commit is allowed on eth1.
- Server preference on this interface is set to maximum (255).
- Assign 2000::20/124 class on fourth interface named eth0. Preference value for this interface should be 0.
- Ignore client with DUID "00001231200adeaaa" in the class 2000::20/124.
- Valid and preferred lifetimes are 1 hour and 30 minutes respectively. T1 and T2 set to 10 minutes and 20 minutes.
- For class 2000::100/124 valid and preferred lifetimes are 2 hours and 1 hour.
- In class 2000::100/124 one client can request up to two addresses.
- Do not assign more than 10 addresses from 2000::20/124 class.
- Client with fe80::200:39ff:fe4b:1abc link-local address should get his static address 2000::2f.
- We shall support DNS and NTP servers on interface 5.
- Log level is set to omit debug messages.

Here's config to do all this stuff:

```
log-level 7
valid-lifetime 3600
preferred-lifetime 1800
T1 600
T2 1200
iface eth0
{
    preference 0
    class
    {
        reject-clients '00001231200adeaaa'
        2000::2f-2000::20 // it's in reverse order, but it works.
    }
}
```

```
                // just a trick.
    }
    class
    {
        accept-only fe80::200:39ff:fe4b:1abc
        pool 2000::2f
    }
}
iface 5
{
    dns-server 2000::123:456,2000::456:1234
    ntp-server 2000::1111:2222
    rapid-commit 1
    preference 255
    class
    {
        pool 2000::fe00-2000::feff
        class-max-lease 10
    }

    class
    {
        valid-lifetime 7200
        preferred-lifetime 3600
        pool 2000::100-2000::10f
        client-max-lease 2
    }
}
```

## 6 Frequently Asked Question

Soon after Dibbler was published, I started to receive questions from users. Some of them were common enough to get into this section.

### 6.1 Common

**Q:** Why client does not configure routing after assigning addresses, so I cannot e.g. ping other hosts?

**A:** It's rather difficult problem. DHCP's job is to obtain address and it exactly does that. To ping any other host, routing should be configured. And this should be done using Router Advertisements. It's kinda odd, but that's the way it was meant to work. If there will be requests from users, I'll think about some enhancements.

**Q:** Dibbler sends some options which have values not recognized by the Ethereal or by other implementations. What's wrong?

**A:** DHCPv6 is a relatively new protocol and additional options are in a specification phase. It means that until standardisation process is over, they do not have any officially assigned numbers. Once standardization process is over (and RFC document is released), this option gets an official number.

There's pretty good chance that different implementors may choose different values for those not-yet officially accepted options. To change those values in Dibbler, you have to modify file `misc/DHCPConst.h` and recompile server or client. See Developer's Guide, section *Option Values* for details.



Currently options with assigned values are: RFC3315: *CLIENT\_ID* , *SERVER\_ID* , *IA\_NA* , *IAADDR* , *OPTION\_REQUEST* , *PREFERENCE* , *ELAPSED* , *STATUS\_CODE* , *RAPID-COMMIT* , *IA\_TA* , *RELAY\_MSG* , *AUTH\_MSG* , *USER\_CLASS* , *VENDOR\_CLASS* , *VENDOR\_OPTS* , *INTERFACE\_ID* , *RECONF\_MSG* , *RECONF\_ACCEPT* ; RFC3319: *SIP\_SERVERS* , *SIP\_DOMAINS* ; RFC3646: *DNS\_RESOLVERS* , *DOMAIN\_LIST* ; RFC3633: *IA\_PD* , *IA\_PREFIX* ; RFC3898: *NIS\_SERVERS* , *NIS+\_SERVERS* , *NIS\_DOMAIN* , *NIS+\_DOMAIN* . Take note that Dibbler does not support all of them.

There are several options which currently does not have values assigned (in parenthesis are numbers used in Dibbler): *NTP\_SERVERS* (40), *TIME\_ZONE* (41), *LIFETIME* (42), *FQDN* (43).

## 6.2 Linux specific

**Q:** I can't run client and server on the same host. What's wrong?

**A:** First of all, running client and server on the same host is just plain meaningless, except testing purposes only. There is a problem with sockets binding. To work around this problem, consult Developer's Guide, Tip section how to compile Dibbler with certain options.

**Q:** After enabling unicast communication, my client fails to send REQUEST messages. What's wrong?

**A:** This is a problem with certain kernels. My limited test capabilities allowed me to conclude that there's problem with 2.4.20 kernel. Everything works fine with 2.6.0 with USAGI patches. Patched kernels with enhanced IPv6 support can be downloaded from <http://www.linux-ipv6.org/>. Please let me know if your kernel works or not.

## 6.3 Windows specific

**Q:** After installing *Advanced Networking Pack* or *Windows XP ServicePack2* my DHCPv6 (or other IPv6 application) stopped working. Is Dibbler XP SP2 compatible?

**A:** In both of this products, there is a IPv6 firewall installed. It is configured by default to reject all incoming IPv6 traffic. You have to disable this firewall. To do so, issue following commands in a console:

## 7 History

Dibbler project was started as master thesis by Tomasz Mrugalski and Marek Senderski on Computer Science faculty on Gdansk University of Technology. Both authors graduated in september 2003 and soon after started their jobs.

During master thesis writing, it came to my attention that there are other DHCPv6 implementations available, but none of them has been named properly. Referring to them was a bit silly: „DHCPv6 published on sourceforge.net has better support than DHCPv6 developed in KAME project, but our DHCPv6 implementation...”. So I have decided that this implementation should have a name. Soon it was named Dibbler after famous CMOT Dibbler from Discworld series by Terry Pratchett.

Sadly, Marek does not have enough free time to develop Dibbler, so his involvement is very limited at this time. However, that does not mean, that this project is abandoned. It is being actively developed by me (Tomek). Keep in mind that I work at full time and do Ph.D. studies, so my free time is also greatly limited.

## 8 Contact

There is an website located at <http://klub.com.pl/dhcpv6>. If you believe you have found a bug, please put it in Bugzilla – it is a bug tracking system located at <http://klub.com.pl/bugzilla>. If you

are not familiar with that kind of system, don't worry. After simple registration, you will be asked for system and Dibbler version you are using and so on. Without feedback from users, author will not be aware of many bugs and so will not be able to fix them. That's why users feedback is very important. You can also send bug report directly using e-mail. Be sure to be as detailed as possible. Please include both server and client log files, both config and xml files. If you are familiar with tcpdump or ethereal, traffic dumps from this programs are also great help.

If you have used Dibbler and it worked ok, this documentation answered all you question and everything is in order (hmmm, wake up, it must be a dream, it isn't reality:), also send a short note to author. He can be contated at thomson(at)klub(dot)com(dot)pl (replace (at) with @ and dot with .). Be sure to include information which country do you live in. It's just author's curiosity to know where Dibbler is being used or tested.

## 9 Thanks and greetings

I would like to send my thanks and greetings to various persons. Without them, Dibbler would not be where is it today.

**Marek Senderski** – He's author of almost half of the Dibbler code. Without his efforts, Dibbler would be simple, long forgotten by now master thesis.

**Jozef Wozniak** – My master thesis' supervisor. He allowed me to see DHCP in a larger scope – as part of total automatisisation process.

**Jacek Swiatowiak** – He's my master thesis consultant. He guided Marek and me to take first steps with DHCPv6 implementation.

**Ania Szulc** – Discworld fan and a great girl, too. She's the one who helped me to decide how to name this yet-untitled DHCPv6 implementation.

**Christian Strauf** – Without his queries and questions, Dibbler would be abadoned half a year or so ago.

**Bartek Gajda** – His interest convinced me that Dibbler is worth the effort to develop it further.

**Artur Binczewski and Maciej Patelczyk** – They both ensured that Dibbler is (and always will be) GNU GPL software. Open source community is grateful.

**Josep Sole** – His mails (directly and indirectly) resulted in various fixes and speeded up 0.2.0 release.