
dyndnsc Documentation

Release 0.4.3

Paul Kremer

Jun 26, 2017

Contents

1	User Guide	1
1.1	Introduction	1
1.2	Installation	1
1.3	Quickstart	2
1.4	Frequently Asked Questions	5
1.5	Community Updates	6
1.6	License	8
2	API Documentation	11
2.1	Developer Interface	11
3	Contributor Guide	13
3.1	Contributing	13
4	Indices and tables	15
	Python Module Index	17

This part of the documentation, which is mostly prose, begins with some background information about Dyndnsc, then focuses on step-by-step instructions for getting the most out of Dyndnsc.

Introduction

Goals

Dyndnsc was born as a minimal module for use in the *ANGEL APP*, a p2p filesystem on top of webdav. At the time, the command line interface was not important. Now that time has passed, the goals are essentially to provide both an easy to use command line tool as well as providing an OK API for developers who want to add dyndns capabilities to their software. Also, providing support for many different ways of detecting and updating IP/DNS records is important.

Installation

This part of the documentation covers the installation of Dyndnsc. The first step to using any software package is getting it properly installed.

Pip

Installing Dyndnsc is simple with pip:

```
$ pip install dyndnsc
```

If using python2 and pip >= 1.5, more arguments are required:

```
$ pip install dyndnsc --allow-external netifaces --allow-unverified netifaces
```

Cheeseshop (PyPI) Mirror

If the Cheeseshop (a.k.a. PyPI) is down, you can also install Dyndnsc from one of the mirrors. [Crate.io](#) is one of them:

```
$ pip install -i http://simple.crate.io/ dyndnsc
```

Get the Code

Dyndnsc is actively developed on GitHub, where the code is [always available](#).

You can either clone the public repository:

```
git clone git://github.com/infothrill/python-dyndnsc.git
```

Download the [tarball](#):

```
$ curl -OL https://github.com/infothrill/python-dyndnsc/tarball/master
```

Or, download the [zipball](#):

```
$ curl -OL https://github.com/infothrill/python-dyndnsc/zipball/master
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

Quickstart

Eager to get started? This page gives a good introduction in how to get started with Dyndnsc. This assumes you already have Dyndnsc installed. If you do not, head over to the [Installation](#) section.

First, make sure that:

- Dyndnsc is *installed*
- Dyndnsc is *up-to-date*

Let's get started with some simple examples.

Command line usage

Dyndnsc exposes all options through the command line interface, however, we do recommend using a configuration file. Here is an example to update an IPv4 record on nsupdate.info with web based IP autodetection:

```
$ dyndnsc --updater-dyndns2 \  
  --updater-dyndns2-hostname test.nsupdate.info \  
  --updater-dyndns2-userid test.nsupdate.info \  
  --updater-dyndns2-password XXXXXXXX \  
  --updater-dyndns2-url https://nsupdate.info/nic/update \  
  --detector-webcheck4 \  
  --detector-webcheck4-url https://ipv4.nsupdate.info/myip \  
  --detector-webcheck4-parser plain
```

Updating an IPv6 address when using Miredo:

```
$ dyndnsc --updater-dyndns2 \
  --updater-dyndns2-hostname test.nsupdate.info \
  --updater-dyndns2-userid test.nsupdate.info \
  --updater-dyndns2-password XXXXXXXX \
  --detector-teredo
```

Updating an IPv6 record on nsupdate.info with interface based IP detection:

```
$ dyndnsc --updater-dyndns2 \
  --updater-dyndns2-hostname test.nsupdate.info \
  --updater-dyndns2-userid test.nsupdate.info \
  --updater-dyndns2-password XXXXXXXX \
  --detector-socket \
  --detector-socket-family INET6
```

Update protocols

Dyndnsc supports several different methods for updating dynamic DNS services:

- **dnsimple** Note: requires python package [dnsimple-dyndns](#) to be installed
- [duckdns](#)
- [dyndns2](#)
- [freedns.afraid.org](#)

A lot of services on the internet offer some form of compatibility, so check against this list. Some of these external services are pre-configured for Dyndnsc as a *preset*, see the section on presets.

Each supported update protocol can be parametrized on the dyndnsc command line using long options starting with ‘--updater-’ followed by the name of the protocol:

```
$ dyndnsc --updater-afraid
$ dyndnsc --updater-dnsimple
$ dyndnsc --updater-duckdns
$ dyndnsc --updater-dyndns2
```

Each of these update protocols supports specific parameters, which might differ from each other. Each of these additional parameters can be specified on the command line by appending them to the long option described above.

Example to specify *token* for updater *duckdns*:

```
$ dyndnsc --updater-duckdns-token 847c0ffb-39bd-326f-b971-bfb3d4e36d7b
```

Detecting the IP

Dyndnsc ships a couple of “detectors” which are capable of finding an IP address through different means.

Detectors may need additional parameters to work properly. Additional parameters can be specified on the command line similarly to the update protocols.

```
$ dyndnsc --detector-iface \
  --detector-iface-iface en0 \
  --detector-iface-family INET
```

```
$ dyndnsc --detector-webcheck4 \  
    --detector-webcheck4-url    http://ipv4.nsupdate.info/myip \  
    --detector-webcheck4-parser plain
```

Some detectors require additional python dependencies:

- *iface*, *teredo* detectors require [netifaces](#) to be installed

Presets

Dyndnsc comes with a list of pre-configured presets. To see all configured presets, you can run

```
$ dyndnsc --list-presets
```

Presets are used to shorten the amount of configuration needed by providing preconfigured parameters. For convenience, *Dyndnsc* ships some built-in presets but this list can be extended by yourself by adding them to the configuration file. Each preset has a section in the ini file called '[preset:NAME]'. See the section on the configuration file to see how to use presets.

Note: Presets can currently only be used in a configuration file. There is currently no support to select a preset from the command line.

Configuration file

Create a config file `test.cfg` with this content (no spaces at the left!):

```
[dyndnsc]  
configs = test_ipv4, test_ipv6  
  
[test_ipv4]  
use_preset = nsupdate.info:ipv4  
updater-hostname = test.nsupdate.info  
updater-userid = test.nsupdate.info  
updater-password = xxxxxxxx  
  
[test_ipv6]  
use_preset = nsupdate.info:ipv6  
updater-hostname = test.nsupdate.info  
updater-userid = test.nsupdate.info  
updater-password = xxxxxxxx
```

Now invoke *dyndnsc* and give this file as configuration:

```
$ dyndnsc --config test.cfg
```

Custom services

If you are using a *dyndns2* compatible service and need to specify the update URL explicitly, you can add the argument `--updater-dyndns2-url`:

```
$ dyndnsc --updater-dyndns2 \  
    --updater-dyndns2-hostname=test.dyndns.com \  
    --updater-dyndns2-userid=bob \  
    --updater-dyndns2-url=http://test.dyndns.com
```



```
--updater-dyndns2-password=fub4r \  
--updater-dyndns2-url=https://dyndns.example.com/nic/update
```

Plugins

Dyndnsc supports plugins which can be notified when a dynamic DNS entry was changed. Currently, only two plugins exist:

- Growl
- OS X Notification

The list of plugins that are deemed compatible and available in your environment will be listed in the command line help. Each plugin command line option starts with ‘-with-’.

Frequently Asked Questions

Python 3 Support?

Yes! In fact, we recommend running on Python3, because a lot of dynamic dns services provide https interfaces, which are best supported in newer python versions from a security and performance point of view.

Here’s a list of Python platforms that are officially supported:

- Python 2.6
- Python 2.7
- Python 3.2
- Python 3.3
- Python 3.4
- PyPy 1.9

Support for Python 2.6 and 3.2 may be dropped at any time.

Is service xyz supported?

To find out wether a certain dynamic dns service is supported by *Dyndnsc*, you can either try to identify the protocol involved and see if it is supported by *Dyndnsc* by looking the output of ‘*dyndnsc -help*’. Or maybe the service in question is already listed in the presets (‘*dyndnsc -list-presets*’).

I get a wrong IPv6 address, why?

If you use the “webcheck6” detector and your system has IPv6 privacy extensions, it’ll result in the temporary IPv6 address that you use to connect to the outside world.

You likely rather want your less private, but static global IPv6 address in DNS and you can determine it using the “socket” detector.

What about error handling of network issues?

Connection errors and timeout errors on the socket level and http level are mostly handled as transient and simply ignored, i.e. updating and/or detecting an IP will fail with a log message but the client should remain active and retry later.

Some errors are not handled gracefully, for example if there is an SSL handshake issue when using a https connection, dyndnsc will typically fail.

Thus, depending on your needs, it might be required to put the dyndnsc client inside a retry loop to run it in a completely unattended way. Don't be fooled by the `-daemon` option, it is available, but the design of the dyndnsc program does not provide longevity guarantees. Feel free to contribute some by sending pull requests!

Community Updates

Tracking development

The best way to track the development of Dyndnsc is through [the GitHub repo](#).

Release history

0.4.3 (June 26th 2017)

- fixed: nsupdate URLs
- fixed: several minor cosmetic issues, mostly testing related

0.4.2 (March 8th 2015)

- added: support for <https://www.duckdns.org>
- fixed: user configuration keys now override built-in presets

0.4.1 (February 16th 2015)

- bugfixes

0.4.0 (February 15th 2015)

- changed (**INCOMPATIBLE**): command line arguments have been drastically adapted to fit different update protocols and detectors
- added: config file support
- added: running against multiple update services in one go using config file
- improved: for python < 3.2, install more dependencies to get SNI support
- improved: the DNS resolution automatically resolves using the same address family (ipv4/A or ipv6/AAAA or any) as the detector configured
- improved: it is now possible to specify arbitrary service URLs for the different updater protocols.
- fixed: naming conventions

- fixed: http connection robustness (i.e. catch more errors and handle them as being transient)
- changed: dependency on netifaces was removed, but if installed, the functionality remains in place
- a bunch of pep8, docstring and documntation updates

0.3.4 (January 3rd 2014)

- added: initial support for dnsimple.com through [dnsimple-dyndns](#)
- added: plugin based desktop notification (growl and OS X notification center)
- changed: for python3.3+, use stdlib 'ipaddress' instead of 'IPy'
- improved: dyndns2 update is now allowed to timeout
- improved: freedns.afraid.org robustness
- improved: webcheck now has an http timeout
- improved: naming conventions in code
- added: initial documentation using sphinx

0.3.3 (December 2nd 2013)

- added: experimental support for <http://freedns.afraid.org>
- added: detecting ipv6 addresses using 'webcheck6' or 'webcheck46'
- fixed: long outstanding state bugs in detector base class
- improved: input validation in Iface detection
- improved: support pytest conventions

0.3.2 (November 16th 2013)

- added: command line option `-debug` to explicitly increase loglevel
- fixed potential race issues in detector base class
- fixed: several typos, test structure, naming conventions, default loglevel
- changed: dynamic importing of detector code

0.3.1 (November 2013)

- added: support for <https://nsupdate.info>
- fixed: automatic installation of 'requests' with setuptools dependencies
- added: more URL sources for 'webcheck' IP detection
- improved: switched optparse to argparse for future-proofing
- fixed: logging initialization warnings
- improved: ship tests with source tarball
- improved: use reStructuredText rather than markdown

0.3 (October 2013)

- moved project to <https://github.com/infothrill/python-dyndnsc>
- added continuous integration tests using <http://travis-ci.org>
- added unittests
- dyndnsc is now a package rather than a single file module
- added more generic observer/subject pattern that can be used for desktop notifications
- removed growl notification
- switched all http related code to the “requests” library
- added <http://www.noip.com>
- removed dyndns.majimoto.net
- dropped support for python <= 2.5 and added support for python 3.2+

0.2.1 (February 2013)

- moved code to git
- minimal PEP8 changes and code restructuring
- provide a makefile to get dependencies using buildout

0.2.0 (February 2010)

- updated IANA reserved IP address space
- Added new IP Detector: running an external command
- Minimal syntax changes based on the 2to3 tool, but remaining compatible with python 2.x

0.1.2 (July 2009)

- Added a couple of documentation files to the source distribution

0.1.1 (September 2008)

- Focus: initial public release

License

Dyndnsc is released under terms of [MIT License](#). This license was chosen to allow inclusion of this software in proprietary and closed systems.

Copyright (c) 2008-2015 Paul Kremer

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

Developer Interface

This part of the documentation should cover all the relevant interfaces of *dyndnsc*.

Main Interface

All of *dyndnsc*' functionality can be accessed by these methods.

class `dyndnsc.DynDnsClient` (*updater=None, detector=None, plugins=None, detect_interval=300*)

This class represents a client to the dynamic dns service.

check ()

Check if the detector changed and call `sync()` accordingly.

If the sleep time has elapsed, this method will see if the attached detector has had a state change and call `sync()` accordingly.

has_state_changed ()

Detect a change either in the offline detector or a difference between the real DNS value and what the online detector last got. This is efficient, since it only generates minimal dns traffic for online detectors and no traffic at all for offline detectors.

Return type boolean

needs_check ()

Check if enough time has elapsed to perform a `check()`.

If this time has elapsed, a state change check through `has_state_changed()` should be performed and eventually a `sync()`.

Return type boolean

needs_sync ()

Check if enough time has elapsed to perform a sync().

A call to sync() should be performed every now and then, no matter what has_state_changed() says. This is really just a safety thing to enforce consistency in case the state gets messed up.

Return type boolean

sync ()

Synchronize the registered IP with the detected IP (if needed).

This can be expensive, mostly depending on the detector, but also because updating the dynamic ip in itself is costly. Therefore, this method should usually only be called on startup or when the state changes.

If you want to contribute to the project, this part of the documentation is for you.

Contributing

Basic method to contribute a change

Dyndnsc is under active development, and contributions are more than welcome!

1. Check for open issues or open a fresh issue to start a discussion around a bug on the [issue tracker](#).
2. Fork [the repository](#) and start making your changes to a new branch.
3. Write a test which shows that the bug was fixed.
4. Send a pull request and bug the maintainer until it gets merged and published. :) Make sure to add yourself to [AUTHORS](#).

Idioms to keep in mind

- keep amount of external dependencies low, i.e. if it can be done using the standard library, do it using the standard library
- do not prefer specific operating systems, i.e. even if we love Linux, we shall not make other suffer from our personal choice
- write unittests

Also, keep these [PEP 20](#) idioms in mind:

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.

4. Complex is better than complicated.
5. Readability counts.

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

d

dyndnsc, [11](#)

C

`check()` (`dyndnsc.DynDnsClient` method), 11

D

`dyndnsc` (module), 11

`DynDnsClient` (class in `dyndnsc`), 11

H

`has_state_changed()` (`dyndnsc.DynDnsClient` method),
11

N

`needs_check()` (`dyndnsc.DynDnsClient` method), 11

`needs_sync()` (`dyndnsc.DynDnsClient` method), 11

P

Python Enhancement Proposals

PEP 20, 13

S

`sync()` (`dyndnsc.DynDnsClient` method), 12