dyndnsc Documentation

Release 0.6.0

Paul Kremer

Feb 21, 2021

CONTENTS

CHAPTER

ONE

USER GUIDE

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.

1.1 Introduction

1.1.1 What is Dyndnsc?

It's a dynamic DNS client. It can detect your IP address in a variety of ways and update DNS records automatically.

1.1.2 Goals

Provide:

- an easy to use command line tool
- an API for developers
- support for a variety of ways to detect IP addresses
- support for a variety of ways to update DNS records

1.2 Installation

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

1.2.1 Pip / pipsi

Installing Dyndnsc is simple with pip:

```
pip install dyndnsc
```

Or, if you prefer a more encapsulated way, use pipsi:

```
pipsi install dyndnsc
```

1.2.2 Docker

Docker images are provided for the following architectures.

x86:

```
docker pull infothrill/dyndnsc-x86-alpine
```

See also https://hub.docker.com/r/infothrill/dyndnsc-x86-alpine/

armhf:

```
docker pull infothrill/dyndnsc-armhf-alpine
```

See also https://hub.docker.com/r/infothrill/dyndnsc-armhf-alpine/

1.2.3 Get the Code

Dyndnsc is developed on GitHub, where the code is available.

You can clone the public repository:

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

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
```

1.3 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.

1.3.1 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
```

1.3.2 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 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
```

1.3.3 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
```

(continues on next page)

(continued from previous page)

```
$ 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

1.3.4 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.

1.3.5 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
```

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

1.3.7 Plugins

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

- dyndnsc-growl
- dyndnsc-macosnotify

The list of plugins that are installed and available in your environment will be listed in the command line help. Each plugin command line option starts with '–with-'.

1.4 Frequently Asked Questions

1.4.1 Python 3 Support?

Yes! In fact, we only support Python3 at this point.

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

- Python 3.6
- Python 3.7
- Python 3.8
- Python 3.9

1.4.2 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').

1.4.3 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.

1.4.4 What about error handling of network issues?

"Hard" errors on the transport level (tcp timeouts, socket erors...) are not handled and will fail the client. In daemon or loop mode, exceptions are caught to keep the client alive (and retries will be issued at a later time).

1.5 Community Updates

1.5.1 Tracking development

The best way to track the development of Dyndnsc is through the GitHub repo.

1.5.2 Release history

0.6.0 (February 21st 2021)

- changed (**INCOMPATIBLE**): dropped support for python 2.7 and python 3.4, 3.5
- added: more presets
- improved: add support for python 3.8, 3.9
- added: docker build automation
- added: *–log-json* command line option, useful when running in docker

0.5.1 (July 7th 2019)

- improved: pin pytest version to version smaller than 5.0.0

0.5.0 (June 25th 2019)

- improved: simplified notification plugin and externalized them using entry_points
- added: WAN IP detection through DNS (detector 'dnswanip')
- improved: replaced built-in daemon code with daemonocle
- switched to pytest for running tests
- changed (**INCOMPATIBLE**): dropped support for python 2.6 and python 3.3
- added: new command line option -v to control verbosity
- improved: infinite loop and daemon stability, diagnostics #57
- improved: updated list of external urls for IP discovery

- improved: install documentation updated
- improved: add many missing docstrings and fixed many code smells
- improved: run flake8 code quality checks in CI
- improved: run check-manifest in CI
- improved: run safety in CI

0.4.4 (December 27th 2017)

- fixed: fixed wheel dependency on python 2.6 and 3.3
- fixed: pep8 related changes, doc fixes

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

1.6 License

Dyndnsc is released under terms of MIT License. This license was chosen explicitly to allow inclusion of this software in proprietary and closed systems.

CHAPTER

TWO

API DOCUMENTATION

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

2.1 API Documentation

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

2.1.1 Main Interface

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

This class represents a client to the dynamic dns service.

Initialize.

Parameters **detect_interval** – amount of time in seconds that can elapse between checks

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 changes in offline detector and real DNS value.

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.

2.1.2 IP Updaters

Afraid

Functionality for interacting with a service compatible with https://freedns.afraid.org/.

Duckdns

Module containing the logic for updating DNS records using the duckdns protocol.

From the duckdns.org website:

https://{DOMAIN}/update?domains={DOMAINLIST}&token={TOKEN}&ip={IP}

where: DOMAIN the service domain DOMAINLIST is either a single domain or a comma separated list of domains TOKEN is the API token for authentication/authorization IP is either the IP or blank for auto-detection

Dyndns2

Module providing functionality to interact with dyndns2 compatible services.

2.1.3 IP Detectors

Command

Module containing logic for command based detectors.

class dyndnsc.detector.command.**IPDetector_Command**(*command=''*, **args*, ***kwargs*)
IPDetector to detect IP address executing shell command/script.

Initialize.

Parameters **command** – string shell command that writes IP address to STDOUT

__init__(*command=''*, **args*, ***kwargs*)
Initialize.

Parameters **command** – string shell command that writes IP address to STDOUT

DNS WAN IP

Module containing logic for DNS WAN IP detection.

See also https://www.cyberciti.biz/faq/how-to-find-my-public-ip-address-from-command-line-on-a-linux/

class dyndnsc.detector.dnswanip.**IPDetector_DnsWanIp**(*family=None*, **args*, ***kwargs*)
Detect the internet visible IP address using publicly available DNS infrastructure.

Initialize.

Parameters **family** – IP address family (default: '' (ANY), also possible: 'INET', 'INET6')

__init__(*family=None*, **args*, ***kwargs*)
Initialize.

Parameters **family** – IP address family (default: '' (ANY), also possible: 'INET', 'INET6')

Interface

Module providing IP detection functionality based on netifaces.

class dyndnsc.detector.iface.**IPDetector_Iface**(*iface=None*, *netmask=None*, *family=None*, **args*, ***kwargs*)
IPDetector to detect an IP address assigned to a local interface.

This is roughly equivalent to using *ifconfig* or *ipconfig*.

Initialize.

Parameters

- **iface** – name of interface
- **family** – IP address family (default: INET, possible: INET6)
- **netmask** – netmask to be matched if multiple IPs on interface (default: none (match all)", example for teredo: "2001:0000::/32")

__init__(*iface=None*, *netmask=None*, *family=None*, **args*, ***kwargs*)
Initialize.

Parameters

- **iface** – name of interface
- **family** – IP address family (default: INET, possible: INET6)
- **netmask** – netmask to be matched if multiple IPs on interface (default: none (match all)", example for teredo: "2001:0000::/32")

Socket

Module containing logic for socket based detectors.

class dyndnsc.detector.socket_ip.**IPDetector_Socket**(*family=None*, **args*, ***kwargs*)
Detect IPs used by the system to communicate with outside world.

Initialize.

Parameters **family** – IP address family (default: INET, possible: INET6)

__init__(*family=None*, **args*, ***kwargs*)
Initialize.

Parameters **family** – IP address family (default: INET, possible: INET6)

Teredo

Module containing logic for teredo based detectors.

class dyndnsc.detector.teredo.**IPDetector_Teredo**(*iface='tun0'*, *netmask='2001:0000::/32'*, **args*, ***kwargs*)

IPDetector to detect a Teredo ipv6 address of a local interface.

Bits 0 to 31 of the ipv6 address are set to the Teredo prefix (normally 2001:0000::/32). This detector only checks the first 16 bits! See http://en.wikipedia.org/wiki/Teredo_tunneling for more information on Teredo.

Inherits IPDetector_Iface and sets default options only.

Initialize.

__init__(*iface='tun0'*, *netmask='2001:0000::/32'*, **args*, ***kwargs*)

Initialize.

Web check

Module containing logic for webcheck based detectors.

class dyndnsc.detector.webcheck.**IPDetectorWebCheck**(**args*, ***kwargs*)

Class to detect an IPv4 address as seen by an online web site.

Return parsable output containing the IP address.

Note: This detection mechanism requires ipv4 connectivity, otherwise it will simply not detect the IP address.

Initialize.

__init__(**args*, ***kwargs*)

Initialize.

CHAPTER

THREE

CONTRIBUTOR GUIDE

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

3.1 Contributing

3.1.1 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.

3.1.2 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

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

Symbols

C

D

H

I

M

N

P

S