Socks5man Documentation

Ricardo van Zutphen

Jul 04, 2018

Contents

Socks5man is a Socks5 management tool and Python library. It enables you to add socks5 servers, run a service that verifies if they are operational, and request these servers in a round-robin fashion by country, city, average connection time, and bandwidth, using the Python library or command line tools.

The library also allows for manual operationality, bandwidth, and connection time tests. A local database is used to lookup country and city information for a host ip.

Socks5man uses a Geolite2 database provided by MaxMind to perform IP to country and city lookups.

Socks5man consists of command line tools and a Python package containing management helpers that can be used in your scripts/programs. These allow you to add, test, list, export, and remove socks5 servers to its database.

Want to use socks5man in your script/program? See:

- *Python modules*

CHAPTER 1

Installation and configuration

This page is a work in progress. More information will be added. Be back soon!

1.1 Installation

It is recommended that socks5man is installed in a virtualenv.

Install Socks5man as follows:

```
$ virtualenv venv
$ source venv/bin/activate
(venv)$ pip install -U socks5man
```

After installing, run Socks5man once to created its working directory (.socks5man) in your user home:

```
(venv)$ socks5man
Usage: socks5man [OPTIONS] COMMAND [ARGS]...

Options:
  -d, --debug  Enable debug logging
  --help       Show this message and exit.

Commands:
  add           Add socks5 server.
  bulk_add      Bulk add socks5 servers from CSV file.
  delete        Remove the specified socks5 servers.
  list          List or export all socks5 servers.
  update-geodb  Update version of the used Maxmind geodb, and...
  verify        Verify if the servers are operational.
```

After installing and running Socks5man once, its working directory will have been created. This directory contains the configuration file (socks5man.conf), the geolite2 database, and the socks5man database. The latter is the database that stores all the socks5 server information.

CHAPTER 2

Command line tools

This page is a work in progress. More information will be added. Be back soon!

CHAPTER 3

Python modules

The following modules and classes should be used to interact with Socks5man.

3.1 Manager module

3.1.1 socks5man.manager

This module contains the class that can be used to interact with Socks5man from within your script/program. It can acquire, list, add, bulk add, and delete socks5 server.

class socks5man.manager.**Manager**

A helper class that should be used to interact with Socks5man. All returned socks5 servers will be returned in a socks5man.socks5.Socks5 wrapper. This allows for direct usage of the retrieved information.

acquire(*country=None*, *country_code=None*, *city=None*, *min_mbps_down=None*, *max_connect_time=None*, *update_usage=True*)

Acquire a socks5 server that was tested to be operational. The returned socks5 server will automatically be marked as used. Acquiring is in a round-robin fashion.

Parameters

- **country** – Country the socks5 server should be in.
- **country_code** – 2-letter country code (ISO 3166-1 alpha-2).
- **city** – City the socks5 server should be in.
- **min_mbps_down** – The minimum average download speed in mbits (float).
- **max_connect_time** – The maximum average connection time in seconds a socks5 server should have (float).
- **update_usage** – Mark retrieved socks5 as used. (bool).

Returns A Socks5 object containing information about the server. None if no matching Socks5 server was found.

Return type *Socks5*

Example

```
>>> from socks5man.manager import Manager
>>> Manager().acquire(country="Germany")
```

add(*host*, *port*, *username=None*, *password=None*, *description=None*)
Add a socks5 server.

Parameters

- **host** – IP or a valid hostname of the socks5 server. Should be unique.
- **port** – Port of the socks5 server (int)
- **username** – Username of the socks5 server (optional)
- **password** – Password for the socks5 server user (optional). Password will be stored in plaintext!
- **description** – Description to store with the socks5 server (optional)

Returns A dictionary containing the provided information, the generated id, the determined country, country code, and city.

Return type dict

Raises Socks5CreationError

Example

```
>>> from socks5man.manager import Manager
>>> Manager().add("example.com", 8456)
    {
        'username': None,
        'city': u'Norwell',
        'host': 'example.com',
        'country_code': u'US',
        'country': u'United States',
        'password': None,
        'port': 8456,
        'id': 1
    }
```

Note: It is only possible to provide both a username and a password. Hostname/IP should be unique. If a socks5 exists with the provided hostname/IP, a Socks5CreationError will be thrown.

---

bulk_add(*socks5_dict_list*, *description=None*)
Bulk add multiple socks5 server. No duplicate checking is done.

Parameters

- **socks5_dict_list** – A list of dictionaries that at a minimum contain the keys and valid values for 'host' and 'port'.
- **description** – A description to be added to all provided servers

Returns The amount of socks5 server that were successfully added

Return type int

Raises Socks5CreationError

Example

```
>>> from socks5man.manager import Manager
>>> Manager().bulk_add([{"host": "example.com", "port": 1234}, {"host":
↪"example.org", "port": 1234}])
2
```

Note: It is only possible to provide both a username and a password for a server. Hostname/IP should be unique. Socks5 servers with invalid hostnames or missing fields will be skipped. Socks5CreationError is raised if no valid servers are in the list.

---

delete(*socks5_id*)
Delete socks5 with given id

Parameters **socks5_id** – A socks5 server id (int)

Example

```
>>> from socks5man.manager import Manager
>>> Manager().delete(1)
```

delete_all()
Remove all socks5 servers from the database

Example

```
>>> from socks5man.manager import Manager
>>> Manager().delete(1)
```

list_socks5(*country=None*, *country_code=None*, *city=None*, *host=None*, *operational=None*)
Retrieve list of existing socks5 servers using the specified filters. This does not mark them as used. It only retrieves a list of matching servers. Returns an empty list if no matches were found. Returns all servers if no filters were provided.

Parameters

- **country** – Country of a socks5 server
- **country_code** – 2-letter country code (ISO 3166-1 alpha-2)
- **city** – City of a socks5 server
- **host** – The host/IP of a socks5 server
- **operational** – Operational or not (bool). Is ignored if value is None

Returns A list of Socks5 objects containing the information of the matching servers.

Return type list

Example

```
>>> from socks5man.manager import Manager
>>> Manager().list_socks5(country="united states")
[
    <Socks5(host=example.com, port=1234, country=United States,␣
↪authenticated=False)>,
```

(continues on next page)

(continued from previous page)

```
    <Socks5(host=example.org, port=1234, country=United States,␣
↪authenticated=False)>
]
```

```
>>> Manager().list_socks5()
[
    <Socks5(host=example.com, port=1234, country=United States,␣
↪authenticated=False)>,
    <Socks5(host=example.org, port=1234, country=United States,␣
↪authenticated=False)>,
    <Socks5(host=example.net, port=1234, country=Germany,␣
↪authenticated=False)>
]
```

3.2 Socks5 module

3.2.1 socks5man.socks5

This module contains the Socks5 class. It is used as a wrapper to contain socks5 server information. The wrapper allows you to verify if a socks5 server is operational, calculate an approximate bandwidth, calculate an approximate connection time, and retrieve all stored server information through properties.

class socks5man.socks5.**Socks5**(*db_socks5*)

Socks5 wrapper class. Retrieve info and verify if socks is operational. Object is initialized with a socks5 database object by the manager class.

verify()

Test if this socks5 can be connected to and retrieve its own IP through the configured IP api. Automatically updates the 'operational' value in the database

Returns True if server is operational, false otherwise

Return type bool

approx_bandwidth()

Calculate an approximate Mbit/s download speed using the file specified in the config to download. Automatically updated in the database

Returns An approximate download speed in Mbit/s

Return type float

measure_connection_time()

Measure the time it takes to connect to the specified connection test URL in the config. Result is automatically stored in the database

Returns An approximate connection time in seconds

Return type float

to_dict()

Dump the underlying database object to a dictionary

Returns A dictionary containing all database values

Return type dict

id
The server id

Return type int

host
The server hostname/IP

Return type str

port
The server port

Return type int

country
The country the server resides in

Return type str

country_code
The (ISO 3166-1 alpha-2) country code of the country the server resides in

Return type str

city
The city the server resides in

Return type str

username
The username for this server

Return type str

password
The password for this server

Return type str

added_on
The date and time this server was added

Return type DateTime

last_use
The date and time this server was last acquired/used. This field is updated when a server is acquired by Manager().acquire()

Return type DateTime

last_check
The date and time this server's operationality was last checked

Return type DateTime

operational
Boolean that tells if the last operationality check was successful or not.

Return type bool

bandwidth
Approximate bandwidth down in Mbit/s. The bandwidth check is only performed if it is enabled in the socks5man.conf

Return type float

connect_time
The approximate connection time in seconds

Return type float

description
The description of this server

Return type str

Python Module Index

S

Index