haproxyadmin Documentation

Release 0.2.1

Pavlos Parissis

Apr 12, 2019

Contents

CHAPTER 1

haproxyadmin

A Python library to manage HAProxy via stats socket.

Contents

1.1 Introduction

haproxyadmin is a Python library for interacting with HAProxy load balancer to perform operations such as enabling/disabling servers. It does that by issuing the appropriate commands over the stats socket provided by HAProxy. It also uses that stats socket for retrieving statistics and changing settings.

HAProxy is a multi-process daemon and each process can only be accessed by a distinct stats socket. There isn't any shared memory for all these processes. That means that if a frontend or backend is managed by more than one processes, you have to find which stats socket you need to send the query/command. This makes the life of a sysadmin a bit difficult as he has to keep track of which stats socket to use for a given object(frontend/backend/server).

haproxyadmin resolves this problem by presenting objects as single entities even when they are managed by multiple processes. It also supports aggregation for various statistics provided by HAProxy. For instance, to report the requests processed by a frontend it queries all processes which manage that frontend and return the sum.

The library works with Python 2.7 and Python 3.6, but for development and testing Python 3.6 is used. The Six Python 2 and 3 Compatibility Library is being used to provide the necessary wrapping over the differences between these 2 major versions of Python.

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> frontends = hap.frontends()
>>> for frontend in frontends:
...     print(frontend.name, frontend.requests, frontend.process_nb)
...
frontend_proc2 0 [2]
haproxy 0 [4, 3, 2, 1]
frontend_proc1 0 [1]
frontend1_proc34 0 [4, 3]
frontend2_proc34 0 [4, 3]
>>>
>>>
>>> backends = hap.backends()
>>> for backend in backends:
...    print(backend.name, backend.requests, backend.process_nb)
...    servers = backend.servers()
...    for server in servers:
...       print(" ", server.name, server.requests)
...
backend_proc2 100 [2]
  bck_proc2_srv4_proc2 25
  bck_proc2_srv3_proc2 25
  bck_proc2_srv1_proc2 25
  bck_proc2_srv2_proc2 25
haproxy 0 [4, 3, 2, 1]
backend1_proc34 16 [4, 3]
  bck1_proc34_srv1 6
  bck_all_srv1 5
  bck1_proc34_srv2 5
backend_proc1 29 [1]
  member2_proc1 14
  member1_proc1 15
  bck_all_srv1 0
backend2_proc34 100 [4, 3]
  bck2_proc34_srv2 97
  bck2_proc34_srv1 2
  bck_all_srv1 1
>>>
```

The documentation of the library is available at http://haproxyadmin.readthedocs.org

1.2 Features

- HAProxy in multi-process mode (nbproc >1)
- UNIX stats socket, no support for querying HTTP statistics page
- Frontend operations

- Backend operations
- Server operations
- ACL operations
- MAP operations
- Aggregation on various statistics
- Change global options for HAProxy

1.3 Installation

Use pip:

```
pip install haproxyadmin
```

From Source:

```
sudo python setup.py install
```

Build (source) RPMs:

```
python setup.py clean --all; python setup.py bdist_rpm
```

Build a source archive for manual installation:

```
python setup.py sdist
```

1.4 Release

1. Bump versions in docs/source/conf.py and haproxyadmin/__init__.py
2. Commit above change with:

   ```
   git commit -av -m'RELEASE 0.1.3 version'
   ```

3. Create a signed tag, pbr will use this for the version number:

   ```
   git tag -s 0.1.3 -m 'bump release'
   ```

4. Create the source distribution archive (the archive will be placed in the **dist** directory):

   ```
   python setup.py sdist
   ```

5. pbr will update ChangeLog file and we want to squeeze them to the previous commit thus we run:

   ```
   git commit -av --amend
   ```

6. Move current tag to the last commit:

   ```
   git tag -fs 0.1.3 -m 'bump release'
   ```

7. Push changes:

```
git push;git push --tags
```

1.5 Development

I would love to hear what other people think about **haproxyadmin** and provide feedback. Please post your comments, bug reports, wishes on my issues page.

1.6 Licensing

Apache 2.0

1.7 Acknowledgement

This program was originally developed for Booking.com. With approval from Booking.com, the code was generalised and published as Open Source on github, for which the author would like to express his gratitude.

1.8 Contacts

Project website: https://github.com/unixsurfer/haproxyadmin

Author: Pavlos Parissis <pavlos.parissis@gmail.com>

CHAPTER 2

User Guide

This part of the documentation covers step-by-step instructions for getting the most out of **haproxyadmin**. It begins by introducing operations related to HAProxy process and then focus on providing the most frequent operations for frontends, backends and servers. In all examples HAProxy is configured with 4 processes, see example HAProxy configuration.

A *HAProxy* object with the name `hap` needs to be created prior running the code mentioned in the following sections:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
```

Warning: Make sure you have appropriate privillage to write in the socket files.

2.1 HAProxy Operations

Get some information about the running processes

```
>>> hap.processids
[871, 870, 869, 868]
>>>
>>> hap.description
'Test server'
>>>
>>> hap.releasedate
'2014/10/31'
>>>
>>> hap.version
'1.5.8'
>>>
>>> hap.uptime
'2d 0h55m09s'
```

(continues on next page)

(continued from previous page)

```
>>>
>>> hap.uptimesec
176112
>>>
>>> hap.nodename
'test.foo.com'
>>>
>>> hap.totalrequests
796
```

Note: `totalrequests` returns the total number of requests that are processed by HAProxy. It counts requests for frontends and backends. Don't forget that a single client request passes HAProxy twice.

Dynamically change the specified global maxconn setting.

```
>>> print(hap.maxconn)
40000
>>> hap.setmaxconn(5000)
True
>>> print(hap.maxconn)
20000
>>>
```

Note: New setting is applied per process and the sum is returned.

Get a list of *Frontend* objects for all frontends

```
>>> frontends = hap.frontends()
>>> for f in frontends:
...     print(f.name)
...
frontend_proc1
haproxy
frontend1_proc34
frontend2_proc34
frontend_proc2
```

Get a *Frontend* object for a single frontend

```
>>> frontend1 = hap.frontend('frontend1_proc34')
>>> frontend1.name, frontend1.process_nb
('frontend1_proc34', [4, 3])
```

Get a list of *Backend* objects for all backends

```
>>> backends = hap.backends()
>>> for b in backends:
...     print(b.name)
...
haproxy
backend1_proc34
backend_proc2
```

(continues on next page)

(continued from previous page)

```
backend_proc1
backend2_proc34
```

Get a *Backend* object for a single backend

```
>>> backend1 = hap.backend('backend1_proc34')
>>> backend1.name, backend1.process_nb
('backend1_proc34', [4, 3])
```

Get a list of *Server* objects for each server

```
>>> servers = hap.servers()
>>> for s in servers:
...     print(s.name, s.backendname)
...
bck1_proc34_srv1 backend1_proc34
bck1_proc34_srv2 backend1_proc34
bck_all_srv1 backend1_proc34
bck_proc2_srv3_proc2 backend_proc2
bck_proc2_srv1_proc2 backend_proc2
bck_proc2_srv4_proc2 backend_proc2
bck_proc2_srv2_proc2 backend_proc2
member1_proc1 backend_proc1
bck_all_srv1 backend_proc1
member2_proc1 backend_proc1
bck2_proc34_srv1 backend2_proc34
bck_all_srv1 backend2_proc34
bck2_proc34_srv2 backend2_proc34
```

Note: if a server is member of more than 1 backends then muliple *Server* objects for the server is returned

Limit the list of server for a specific pool

```
>>> servers = hap.servers(backend='backend1_proc34')
>>> for s in servers:
...     print(s.name, s.backendname)
...
bck1_proc34_srv1 backend1_proc34
bck1_proc34_srv2 backend1_proc34
bck_all_srv1 backend1_proc34
```

Work on specific server across all backends

```
>>> s1 = hap.server(hostname='bck_all_srv1')
>>> for x in s1:
...     print(x.name, x.backendname, x.status)
...     x.setstate(haproxy.STATE_DISABLE)
...     print(x.status)
...
bck_all_srv1 backend1_proc34 DOWN
True
MAINT
bck_all_srv1 backend_proc1 DOWN
True
MAINT
```

(continues on next page)

(continued from previous page)

```
bck_all_srv1 backend2_proc34 no check
True
MAINT
```

Examples for ACLs

```
>>> from pprint import pprint
>>> pprint(hap.show_acl())
['# id (file) description',
"0 (/etc/haproxy/wl_stats) pattern loaded from file '/etc/haproxy/wl_stats' "
"used by acl at file '/etc/haproxy/haproxy.cfg' line 53",
"1 () acl 'src' file '/etc/haproxy/haproxy.cfg' line 53",
"3 () acl 'ssl_fc' file '/etc/haproxy/haproxy.cfg' line 85",
'4 (/etc/haproxy/bl_frontend) pattern loaded from file '
"'/etc/haproxy/bl_frontend' used by acl at file '/etc/haproxy/haproxy.cfg' "
'line 97',
"5 () acl 'src' file '/etc/haproxy/haproxy.cfg' line 97",
"6 () acl 'path_beg' file '/etc/haproxy/haproxy.cfg' line 99",
"7 () acl 'req.cook' file '/etc/haproxy/haproxy.cfg' line 114",
"8 () acl 'req.cook' file '/etc/haproxy/haproxy.cfg' line 115",
"9 () acl 'req.cook' file '/etc/haproxy/haproxy.cfg' line 116",
'']
>>> hap.show_acl(6)
['0x12ea940 /static/css/', '']
>>> hap.add_acl(6, '/foobar')
True
>>> hap.show_acl(6)
['0x12ea940 /static/css/', '0x13a38b0 /foobar', '']
>>> hap.add_acl(6, '/foobar')
True
>>> hap.show_acl(6)
['0x12ea940 /static/css/', '0x13a38b0 /foobar', '0x13a3930 /foobar', '']
>>> hap.del_acl(6, '/foobar')
True
>>> hap.show_acl(6)
['0x12ea8a0 /static/js/', '0x12ea940 /static/css/', '']
```

Examples for MAPs

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_map(map=6)
['# id (file) description',
"0 (/etc/haproxy/v-m1-bk) pattern loaded ...... line 82",
'']
>>> hap.show_map(0)
['0x1a78ab0 0 www.foo.com-0', '0x1a78b20 1 www.foo.com-1', '']
```

Manage MAPs

```
>>> hap.show_map(0)
['0x1a78b20 1 www.foo.com-1', '']
>>> hap.add_map(0, '9', 'foo')
True
>>> hap.show_map(0)
['0x1a78b20 1 www.foo.com-1', '0x1b15c80 9 foo', '']
```

```
>>> hap.show_map(0)
['0x1b15cd0 9 foo', '0x1a78980 11 bar', '']
>>> hap.del_map(0, '0x1b15cd0')
True
>>> hap.show_map(0)
['0x1a78980 11 bar', '']
>>> hap.add_map(0, '22', 'bar22')
True
>>> hap.show_map(0)
['0x1a78980 11 bar', '0x1b15c00 22 bar22', '']
>>> hap.del_map(0, '22')
True
>>> hap.show_map(0)
['0x1a78980 11 bar', '']
```

2.2 Frontend Operations

A quick way to check if a certain frontend exists

```
>>> frontends = hap.frontends()
>>> if 'frontend2_proc34' in frontends:
...     print('have it')
...
have it
>>> if 'frontend2_proc34foo' in frontends:
...     print('have it')
...
>>>
```

Change maximum connections to all frontends

```
>>> frontends = hap.frontends()
>>> for f in frontends:
...     print(f.maxconn, f.name)
...     f.setmaxconn(10000)
...     print(f.maxconn, f.name)
...     print('---------------')
...
3000 haproxy-stats2
True
10000 haproxy-stats2
---------------
6000 frontend1_proc34
True
20000 frontend1_proc34
---------------
6000 frontend2_proc34
True
20000 frontend2_proc34
---------------
3000 frontend_proc2
True
10000 frontend_proc2
---------------
3000 haproxy-stats
```

(continues on next page)

(continued from previous page)

```
True
10000 haproxy-stats
---------------
3000 haproxy-stats3
True
10000 haproxy-stats3
---------------
3000 haproxy-stats4
True
10000 haproxy-stats4
---------------
3000 frontend_proc1
True
10000 frontend_proc1
---------------
```

Do the same only on specific frontend

```
>>> frontend = hap.frontend('frontend1_proc34')
>>> frontend.maxconn
20000
>>> frontend.setmaxconn(50000)
True
>>> frontend.maxconn
100000
```

Disable and enable a frontend

```
>>> frontend = hap.frontend('frontend1_proc34')
>>> frontend.status
'OPEN'
>>> frontend.disable()
True
>>> frontend.status
'STOP'
>>> frontend.enable()
True
>>> frontend.status
'OPEN'
```

Shutdown a frontend

```
>>> frontend.shutdown()
True
```

Warning: HAProxy removes from the running configuration the frontend, so further operations on the frontend will return an error.

```
>>> frontend.status
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/..ages/haproxyadmin/frontend.py", line 243, in status
    'status')
File "/...ages/haproxyadmin/utils.py", line 168, in cmd_across_all_procs
```

(continues on next page)

(continued from previous page)

```
    (getattr(obj, 'process_nb'), getattr(obj, method)(*arg))
File "/...ages/haproxyadmin/internal.py", line 210, in metric
    getattr(self.hap_process.frontends_stats()[self.name], name))
KeyError: 'frontend1_proc34'
```

Retrieve various statistics

```
>>> frontend = hap.frontend('frontend2_proc34')
>>> for m in FRONTEND_METRICS:
...     print("name {} value {}".format(m, frontend.metric(m)))
...
name bin value 380
name bout value 1065
name comp_byp value 0
name comp_in value 0
name comp_out value 0
name comp_rsp value 0
name dreq value 0
name dresp value 0
name ereq value 0
name hrsp_1xx value 0
name hrsp_2xx value 0
name hrsp_3xx value 0
name hrsp_4xx value 0
name hrsp_5xx value 5
name hrsp_other value 0
name rate value 0
name rate_lim value 200000
name rate_max value 2
name req_rate value 0
name req_rate_max value 2
name req_tot value 5
name scur value 0
name slim value 20000
name smax value 3
name stot value 5
>>>
>>> frontend.process_nb
[4, 3]
>>> frontend.requests_per_process()
[(4, 2), (3, 3)]
>>> frontend.requests
5
>>>
```

Note: `requests` returns HTTP requests that are processed by the frontend. If the frontend is in TCP mode the number will be always 0 and *stot* metric should be used to retrieve the number of TCP requests processsed.

Read *Frontend* class for more information.

2.3 Backend Operations

A quick way to check if a certain backend exists

```
>>> backends = hap.backends()
>>> if 'backend1_proc34' in backends:
...     print('have it')
...
have it
>>> if 'backend1_proc34foo' in backends:
...     print('have it')
...
>>>
```

Retrieve various statistics

```
>>> backend = hap.backend('backend1_proc34')
>>> for m in BACKEND_METRICS:
...     print("name {} value {}".format(m, backend.metric(m)))
...
name act value 3
name bck value 0
name bin value 0
name bout value 0
name chkdown value 2
name cli_abrt value 0
name comp_byp value 0
name comp_in value 0
name comp_out value 0
name comp_rsp value 0
name ctime value 0
name downtime value 8237
name dreq value 0
name dresp value 0
name econ value 0
name eresp value 0
name hrsp_1xx value 0
name hrsp_2xx value 0
name hrsp_3xx value 0
name hrsp_4xx value 0
name hrsp_5xx value 0
name hrsp_other value 0
name lastchg value 11373
name lastsess value -1
name lbtot value 0
name qcur value 0
name qmax value 0
name qtime value 0
name rate value 0
name rate_max value 0
name rtime value 0
name scur value 0
name slim value 200000
name smax value 0
name srv_abrt value 0
name stot value 0
name ttime value 0
name weight value 3
name wredis value 0
name wretr value 0
>>>
```

(continues on next page)

(continued from previous page)

```
>>>
>>> backend.process_nb
[4, 3]
>>> backend.requests_per_process()
[(4, 2), (3, 3)]
>>> backend.requests
5
>>>
```

Get all servers in across all backends

```
>>> for backend in backends:
>>> backends = hap.backends()
...     print(backend.name, backend.requests, backend.process_nb)
...     servers = backend.servers()
...     for server in servers:
...         print(" ", server.name, server.requests)
...
backend_proc2 100 [2]
  bck_proc2_srv4_proc2 25
  bck_proc2_srv3_proc2 25
  bck_proc2_srv1_proc2 25
  bck_proc2_srv2_proc2 25
haproxy 0 [4, 3, 2, 1]
backend1_proc34 16 [4, 3]
  bck1_proc34_srv1 6
  bck_all_srv1 5
  bck1_proc34_srv2 5
backend_proc1 29 [1]
  member2_proc1 14
  member1_proc1 15
  bck_all_srv1 0
backend2_proc34 100 [4, 3]
  bck2_proc34_srv2 97
  bck2_proc34_srv1 2
  bck_all_srv1 1
>>>
```

Get servers of a specific backend

```
>>> backend = hap.backend('backend1_proc34')
>>> for s in backend.servers():
...     print(s.name, s.status, s.weight)
...
bck1_proc34_srv2 UP 1
bck_all_srv1 UP 1
bck1_proc34_srv1 UP 1
>>>
```

Get a specific server from a backend

```
>>> s1 = backend.server('bck1_proc34_srv2')
>>> s1.name, s1.backendname, s1.status, s1.requests, s1.weight
('bck1_proc34_srv2', 'backend1_proc34', 'UP', 9, 1)
```

Read *Backend* class for more information.

2.4 Server Operations

A quick way to check if a certain server exists

```
>>> servers = hap.servers()
>>> if 'bck_all_srv1' in servers:
...     print("have it")
...
have it
>>> if 'bck_all_srv1foo' in servers:
...     print("have it")
...
>>>
```

Retrieve various statistics

```
>>> backend = hap.backend('backend1_proc34')
>>> for server in backend.servers():
...     print(server.name)
...     for m in SERVER_METRICS:
...         print("name {} value {}".format(m, server.metric(m)))
...     print("-----------")
...
bck1_proc34_srv2
name qcur value 0
name qmax value 0
name scur value 0
name smax value 0
name stot value 0
name bin value 0
name bout value 0
name dresp value 0
name econ value 0
name eresp value 0
name wretr value 0
name wredis value 0
name weight value 1
name act value 1
name bck value 0
name chkfail value 6
name chkdown value 4
name lastchg value 39464
name downtime value 47702
name qlimit value 0
name throttle value 0
name lbtot value 0
name rate value 0
name rate_max value 0
name check_duration value 5001
name hrsp_1xx value 0
name hrsp_2xx value 0
name hrsp_3xx value 0
name hrsp_4xx value 0
name hrsp_5xx value 0
name hrsp_other value 0
name cli_abrt value 0
name srv_abrt value 0
```

(continues on next page)

(continued from previous page)

```
name lastsess value -1
name qtime value 0
name ctime value 0
name rtime value 0
name ttime value 0
-----------
bck1_proc34_srv1
name qcur value 0
name qmax value 0
name scur value 0
name smax value 0
name stot value 0
name bin value 0
name bout value 0
name dresp value 0
name econ value 0
name eresp value 0
name wretr value 0
name wredis value 0
name weight value 1
name act value 1
name bck value 0
name chkfail value 6
name chkdown value 4
name lastchg value 39464
name downtime value 47702
name qlimit value 0
name throttle value 0
name lbtot value 0
name rate value 0
name rate_max value 0
name check_duration value 5001
name hrsp_1xx value 0
name hrsp_2xx value 0
name hrsp_3xx value 0
name hrsp_4xx value 0
name hrsp_5xx value 0
name hrsp_other value 0
name cli_abrt value 0
name srv_abrt value 0
name lastsess value -1
name qtime value 0
name ctime value 0
name rtime value 0
name ttime value 0
-----------
bck_all_srv1
name qcur value 0
name qmax value 0
name scur value 0
name smax value 0
name stot value 0
name bin value 0
name bout value 0
name dresp value 0
name econ value 0
name eresp value 0
```

(continues on next page)

(continued from previous page)

```
name wretr value 0
name wredis value 0
name weight value 1
name act value 1
name bck value 0
name chkfail value 6
name chkdown value 4
name lastchg value 39462
name downtime value 47700
name qlimit value 0
name throttle value 0
name lbtot value 0
name rate value 0
name rate_max value 0
name check_duration value 5001
name hrsp_1xx value 0
name hrsp_2xx value 0
name hrsp_3xx value 0
name hrsp_4xx value 0
name hrsp_5xx value 0
name hrsp_other value 0
name cli_abrt value 0
name srv_abrt value 0
name lastsess value -1
name qtime value 0
name ctime value 0
name rtime value 0
name ttime value 0
-----------
>>>
```

Change weight of server in a backend

```
>>> backend = hap.backend('backend1_proc34')
>>> server = backend.server('bck_all_srv1')
>>> server.weight
100
>>> server.setweight('20%')
True
>>> server.weight
20
>>> server.setweight(58)
True
>>> server.weight
58
```

Note: If the value ends with the '%' sign, then the new weight will be relative to the initially configured weight. Absolute weights are permitted between 0 and 256.

or across all backends

```
>>> server_per_backend = hap.server('bck_all_srv1')
>>> for server in server_per_backend:
...     print(server.backendname, server.weight)
...     server.setweight(8)
```

(continues on next page)

(continued from previous page)

```
...     print(server.backendname, server.weight)
...
backend2_proc34 1
True
backend2_proc34 8
backend1_proc34 0
True
backend1_proc34 8
backend_proc1 100
True
backend_proc1 8
>>>
```

Terminate all the sessions attached to the specified server.

```
>>> backend = hap.backend('backend1_proc34')
>>> server = backend.server('bck_all_srv1')
>>> server.metric('scur')
8
>>> server.shutdown()
True
>>> server.metric('scur')
0
```

Disable a server in a backend

```
>>> server = hap.server('member_bkall', backend='backend_proc1')[0]
>>> server.setstate(haproxy.STATE_DISABLE)
True
>>> server.status
'MAINT'
>>> server.setstate(haproxy.STATE_ENABLE)
True
>>> server.status
'no check'
```

Get status of server

```
>>> backend = hap.backend('backend1_proc34')
>>> server = backend.server('bck_all_srv1')
>>> server.last_agent_check
''
>>> server.check_status
'L4TOUT'
>>> server.check_
server.check_code    server.check_status
>>> server.check_code
''
>>> server.status
'DOWN'
>>>
```

Read *Server* class for more information.

CHAPTER 3

Developer Interface

This part of the documentation covers all the available interfaces of haproxyadmin package. Public and internal interfaces are described.

HAProxy, *Frontend*, *Backend* and *server* classes are the main 4 public interfaces. These classes provide methods to run various operations. HAProxy provides a several statistics which can be retrieved by calling `metric()`, see HAProxy statistics for the full list of statistics.

`haproxyadmin.internal` module provides a set of classes that are not meant for external use.

3.1 haproxyadmin.haproxy

This module implements the main haproxyadmin API.

class haproxyadmin.haproxy.**HAProxy**(*socket_dir=None*, *socket_file=None*, *retry=2*, *retry_interval=2*)

Build a user-created *HAProxy* object for HAProxy.

This is the main class to interact with HAProxy and provides methods to create objects for managing frontends, backends and servers. It also provides an interface to interact with HAProxy as a way to retrieve settings/statistics but also change settings.

ACLs and MAPs are also managed by *HAProxy* class.

Parameters

- **socket_dir** (`string`) – a directory with HAProxy stats files.
- **socket_file** (`string`) – absolute path of HAProxy stats file.
- **retry** (`integer` or `None`) – number of times to retry to open a UNIX socket connection after a failure occurred, possible values
  - None => don't retry
  - 0 => retry indefinitely
  - 1..N => times to retry

- **retry_interval** (integer) – sleep time between the retries.

Returns a user-created *HAProxy* object.

Return type *HAProxy*

add_acl(*acl*, *pattern*)
Add an entry into the acl.

Parameters

- **acl** (integer or a file path passed as string) – acl id or a file.
- **pattern** (string) – entry to add.

Returns True if command succeeds otherwise False

Return type bool

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_acl(acl=4)
['0x23181c0 /static/css/']
>>> hap.add_acl(acl=4, pattern='/foo/' )
True
>>> hap.show_acl(acl=4)
['0x23181c0 /static/css/', '0x238f790 /foo/']
```

add_map(*mapid*, *key*, *value*)
Add an entry into the map.

Parameters

- **mapid** (integer or a file path passed as string) – map id or a file.
- **key** (string) – key to add.
- **value** (string) – Value assciated to the key.

Returns True if command succeeds otherwise False.

Return type bool

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_map(0)
['0x1a78b20 1 www.foo.com-1']
>>> hap.add_map(0, '9', 'foo')
True
>>> hap.show_map(0)
['0x1a78b20 1 www.foo.com-1', '0x1b15c80 9 foo']
```

backend(*name*)
Build a *Backend* object.

Parameters **name** (string) – backend name to look up.

Raises :class::*ValueError* when backend isn't found or more than 1 backend is found.

backends(*name=None*)
Build a list of *Backend*

Parameters **name** (`string`) – (optional) backend name to look up.

Returns list of *Backend*.

Return type `list`

clear_acl (*acl*)
Remove all entries from a acl.

Parameters **acl** (`integer` or a file path passed as `string`) – acl id or a file.

Returns True if command succeeds otherwise False

Return type bool

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.clear_acl(acl=4)
True
>>> hap.clear_acl(acl='/etc/haproxy/bl_frontend')
True
```

clear_map (*mapid*)
Remove all entries from a mapid.

Parameters **mapid** (`integer` or a file path passed as `string`) – map id or a file

Returns `True` if command succeeds otherwise `False`

Return type `bool`

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.clear_map(0)
True
>>> hap.clear_map(mapid='/etc/haproxy/bl_frontend')
True
```

clearcounters (*all=False*)
Clear the max values of the statistics counters.

When `all` is set to `True` clears all statistics counters in each proxy (frontend & backend) and in each server. This has the same effect as restarting.

Parameters **all** (`bool`) – (optional) clear all statistics counters.

Returns `True` if command succeeds otherwise `False`.

Return type `bool`

command (*cmd*)
Send a command to haproxy process.

This allows a user to send any kind of command to haproxy. We **do not* perfom any sanitization on input and on output.

Parameters **cmd** (`string`) – a command to send to haproxy process.

Returns list of 2-item tuple

1. HAProxy process number

2. what the method returned

Return type `list`

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.command('show stats')
['0x23181c0 /static/css/']
>>> hap.add_acl(acl=4, pattern='/foo/' )
True
>>> hap.show_acl(acl=4)
['0x23181c0 /static/css/', '0x238f790 /foo/']
```

del_acl (*acl*, *key*)

Delete all the acl entries from the acl corresponding to the key.

Parameters

- **acl** (`integer` or a file path passed as `string`) – acl id or a file
- **key** (`string`) – key to delete.

Returns `True` if command succeeds otherwise `False`.

Return type `bool`

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_acl(acl=4)
['0x23181c0 /static/css/', '0x238f790 /foo/', '0x238f810 /bar/']
>>> hap.del_acl(acl=4, key='/static/css/')
True
>>> hap.show_acl(acl=4)
['0x238f790 /foo/', '0x238f810 /bar/']
>>> hap.del_acl(acl=4, key='0x238f790')
True
>>> hap.show_acl(acl=4)
['0x238f810 /bar/']
```

del_map (*mapid*, *key*)

Delete all the map entries from the map corresponding to the key.

Parameters

- **mapid** (`integer` or a file path passed as `string`.) – map id or a file.
- **key** (`string`) – key to delete

Returns `True` if command succeeds otherwise `False`.

Return type `bool`

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_map(0)
['0x1b15cd0 9 foo', '0x1a78980 11 bar']
```

(continues on next page)

(continued from previous page)

```
>>> hap.del_map(0, '0x1b15cd0')
True
>>> hap.show_map(0)
['0x1a78980 11 bar']
>>> hap.add_map(0, '22', 'bar22')
True
>>> hap.show_map(0)
['0x1a78980 11 bar', '0x1b15c00 22 bar22']
>>> hap.del_map(0, '22')
True
>>> hap.show_map(0)
['0x1a78980 11 bar']
```

errors (*iid=None*)
Dump last known request and response errors.

If <iid> is specified, the limit the dump to errors concerning either frontend or backend whose ID is <iid>.

Parameters **iid** (*integer*) – (optional) ID of frontend or backend.

Returns

A list of tuples of errors per process.

1. process number
2. `list` of errors

Return type `list`

frontend (*name*)
Build a *Frontend* object.

Parameters **name** (`string`) – frontend name to look up.

Returns a *Frontend* object for the frontend.

Return type *Frontend*

Raises :class::*ValueError* when frontend isn't found or more than 1 frontend is found.

frontends (*name=None*)
Build a list of *Frontend*

Parameters **name** (`string`) – (optional) frontend name to look up.

Returns list of *Frontend*.

Return type `list`

get_acl (*acl*, *value*)
Lookup the value in the ACL.

Parameters

- **acl** (`integer` or a file path passed as `string`) – acl id or a file.
- **value** (`string`) – value to lookup

Returns matching patterns associated with ACL.

Return type `string`

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_acl(acl=4)
['0x2318120 /static/js/', '0x23181c0 /static/css/']
>>> hap.get_acl(acl=4, value='/foo')
'type=beg, case=sensitive, match=no'
>>> hap.get_acl(acl=4, value='/static/js/')
'type=beg, case=sensitive, match=yes, idx=tree, pattern="/static/js/"'
```

get_map(*mapid*, *value*)
Lookup the value in the map.

Parameters

- **mapid** (integer or a file path passed as string) – map id or a file.
- **value** (string) – value to lookup.

Returns matching patterns associated with map.

Return type string

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_map(0)
['0x1a78980 11 new2', '0x1b15c00 22 0']
>>> hap.get_map(0, '11')
'type=str, case=sensitive, found=yes, idx=tree, key="11", value="new2", type=
↪"str"'
>>> hap.get_map(0, '10')
'type=str, case=sensitive, found=no'
```

info()
Dump info about haproxy stats on current process.

Returns A list of dict for each process.

Return type list

metric(*name*)
Return the value of a metric.

Performs a calculation on the metric across all HAProxy processes. The type of calculation is either sum or avg and defined in *haproxyadmin.utils.METRICS_SUM* and *haproxyadmin.utils.METRICS_AVG*.

Parameters **name** (any of haproxyadmin.haproxy.HAPROXY_METRICS) – metric name to retrieve

Returns value of the metric

Return type integer

Raise ValueError when a given metric is not found

server(*hostname*, *backend=None*)
Build Server <haproxyadmin.server.Server> for a server. objects for the given server.

If backend specified then lookup is limited to that backend.

Note: If a server is member of more than 1 backend then muliple objects for the same server is returned.

Parameters

- **hostname** (string) – servername to look for.
- **backend** (string) – (optional) backend name to look in.

Returns a list of *Server* objects.

Return type list

servers (*backend=None*)

Build *Server* for each server.

If backend specified then lookup is limited to that backend.

Parameters **backend** (string) – (optional) backend name.

Returns A list of Server objects

Return type list.

set_map (*mapid*, *key*, *value*)

Modify the value corresponding to each key in a map.

mapid is the #<id> or <file> returned by *show_map*.

Parameters

- **mapid** (integer or a file path passed as string) – map id or a file.
- **key** (string) – key id
- **value** (string) – value to set for the key.

Returns True if command succeeds otherwise False.

Return type bool

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_map(0)
['0x1a78980 11 9', '0x1b15c00 22 0']
>>> hap.set_map(0, '11', 'new')
True
>>> hap.show_map(0)
['0x1a78980 11 new', '0x1b15c00 22 0']
>>> hap.set_map(0, '0x1a78980', 'new2')
True
>>> hap.show_map(0)
['0x1a78980 11 new2', '0x1b15c00 22 0']
```

setmaxconn (*value*)

Set maximum connection to the frontend.

Parameters **value** (integer) – value to set.

Returns True if command succeeds otherwise False.

Return type bool

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.setmaxconn(5555)
True
```

setratelimitconn(*value*)
Set process-wide connection rate limit.

Parameters **value** (integer) – rate connection limit.

Returns True if command succeeds otherwise False.

Return type bool

Raises ValueError if value is not an integer.

setratelimitsess(*value*)
Set process-wide session rate limit.

Parameters **value** (integer) – rate session limit.

Returns True if command succeeds otherwise False.

Return type bool

Raises ValueError if value is not an integer.

setratelimitsslsess(*value*)
Set process-wide ssl session rate limit.

Parameters **value** (integer) – rate ssl session limit.

Returns True if command succeeds otherwise False.

Return type bool

Raises ValueError if value is not an integer.

show_acl(*aclid=None*)
Dump info about acls.

Without argument, the list of all available acls is returned. If a aclid is specified, its contents are dumped.

Parameters **aclid** (integer or a file path passed as string) – (optional) acl id or a file

Returns a list with the acls

Return type list

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_acl(aclid=6)
['0x1d09730 ver%3A27%3Bvar%3A0']
>>> hap.show_acl()
['# id (file) description',
"1 () acl 'ssl_fc' file '/etc/haproxy/haproxy.cfg' line 83",
"2 () acl 'src' file '/etc/haproxy/haproxy.cfg' line 95",
"3 () acl 'path_beg' file '/etc/haproxy/haproxy.cfg' line 97",
]
```

show_map (*mapid=None*)
Dump info about maps.

Without argument, the list of all available maps is returned. If a mapid is specified, its contents are dumped.

Parameters **mapid** (integer or a file path passed as string) – (optional) map id or a file.

Returns a list with the maps.

Return type list

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.show_map()
['# id (file) description',
 "0 (/etc/haproxy/v-m1-bk) pattern loaded ...... line 82",
 ]
>>> hap.show_map(mapid=0)
['0x1a78ab0 0 www.foo.com-0', '0x1a78b20 1 www.foo.com-1']
```

description
Return description of HAProxy

Return type string

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.description
'test'
```

maxconn
Return the sum of configured maximum connection allowed for HAProxy.

Return type integer

nodename
Return nodename of HAProxy

Return type string

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.nodename
'test.foo.com'
```

processids
Return the process IDs of all HAProxy processes.

Return type list

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.processids
[22029, 22028, 22027, 22026]
```

ratelimitconn
Return the process-wide connection rate limit.

ratelimitsess
Return the process-wide session rate limit.

ratelimitsslsess
Return the process-wide ssl session rate limit.

releasedate
Return release date of HAProxy

Return type `string`

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.releasedate
'2014/10/31'
```

requests
Return total requests processed by all frontends.

Return type `integer`

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.requests
457
```

totalrequests
Return total cumulative number of requests processed by all processes.

Return type `integer`

Note: This is the total number of requests that are processed by HAProxy. It counts requests for frontends and backends. Don't forget that a single client request passes HAProxy twice.

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.totalrequests
457
```

uptime
Return uptime of HAProxy process

Return type string

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.uptime
'4d 0h16m26s'
```

uptimesec
Return uptime of HAProxy process in seconds

Return type integer

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.uptimesec
346588
```

version
Return version of HAProxy

Return type string

Usage::

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> hap.version
'1.5.8'
```

3.2 haproxyadmin.frontend

This module provides the *Frontend* class. This class can be used to run operations on a frontend and retrieve statistics.

class haproxyadmin.frontend.**Frontend**(*frontend_per_proc*)
Build a user-created *Frontend* for a single frontend.

Parameters **frontend_per_proc** (list) – list of *_Frontend* objects.

Return type a *Frontend*.

disable()
Disable frontend.

Parameters **die** (bool) – control the handling of errors.

Returns True if frontend is disabled otherwise False.

Return type bool

Raise If die is True *haproxyadmin.exceptions.CommandFailed* or *haproxyadmin.exceptions.MultipleCommandResults* is raised when something bad happens otherwise returns False.

enable()
Enable frontend.

Parameters **die** (bool) – control the handling of errors.

Returns True if frontend is enabled otherwise False.

Return type bool

Raise If `die` is `True` *haproxyadmin.exceptions.CommandFailed* or *haproxyadmin.exceptions.MultipleCommandResults* is raised when something bad happens otherwise returns `False`.

metric(*name*)

Return the value of a metric.

Performs a calculation on the metric across all HAProxy processes. The type of calculation is either sum or avg and defined in *haproxyadmin.utils.METRICS_SUM* and *haproxyadmin.utils.METRICS_AVG*.

Parameters **name** (any of `haproxyadmin.haproxy.FRONTEND_METRICS`) – metric name to retrieve

Returns value of the metric

Return type `integer`

Raise `ValueError` when a given metric is not found

requests_per_process()

Return the number of requests for the frontend per process.

Returns

a list of tuples with 2 elements

1. process number of HAProxy
2. requests

Return type `list`

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> frontend = hap.frontend('frontend2_proc34')
>>> frontend.requests_per_process()
[(4, 2), (3, 3)]
```

setmaxconn(*value*)

Set maximum connection to the frontend.

Parameters

- **die** (`bool`) – control the handling of errors.
- **value** (`integer`) – max connection value.

Returns `True` if value was set.

Return type `bool`

Raise If `die` is `True` *haproxyadmin.exceptions.CommandFailed* or *haproxyadmin.exceptions.MultipleCommandResults* is raised when something bad happens otherwise returns `False`.

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> frontend = hap.frontend('frontend1_proc34')
>>> frontend.maxconn
```

(continues on next page)

(continued from previous page)

```
>>> frontend.setmaxconn(50000)
True
>>> frontend.maxconn
100000
```

shutdown()
Disable the frontend.

> **Warning:** HAProxy removes from the running configuration a frontend, so further operations on the frontend will return an error.

Return type bool

stats_per_process()
Return all stats of the frontend per process.

Returns

a list of tuples with 2 elements

1. process number
2. a dict with all stats

Return type list

iid
Return the unique proxy ID of the frontend.

> **Note:** Because proxy ID is the same across all processes, we return the proxy ID from the 1st process.

Return type int

maxconn
Return the configured maximum connection allowed for frontend.

Return type integer

name
Return the name of the frontend.

Return type string

process_nb
Return a list of process number in which frontend is configured.

Return type list

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> frontend = hap.frontend('frontend2_proc34')
>>> frontend.process_nb
[4, 3]
```

requests
Return the number of requests.

Return type integer

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> frontend = hap.frontend('frontend2_proc34')
>>> frontend.requests
5
```

status
Return the status of the frontend.

Return type string

Raise IncosistentData exception if status is different per process

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> frontend = hap.frontend('frontend2_proc34')
>>> frontend.status
'OPEN'
```

3.3 haproxyadmin.backend

This module provides the *Backend* class which allows to run operation for a backend.

class haproxyadmin.backend.**Backend**(*backend_per_proc*)
Build a user-created *Backend* for a single backend.

Parameters **backend_per_proc** (list) – list of *_Backend* objects.

Return type a *Backend*.

metric(*name*)
Return the value of a metric.

Performs a calculation on the metric across all HAProxy processes. The type of calculation is either sum or avg and defined in utils.METRICS_SUM and utils.METRICS_AVG.

Parameters **name** (string) – Name of the metric, any of BACKEND_METRICS

Returns Value of the metric after the appropriate calculation has been performed.

Return type number, either integer or float.

Raise ValueError when a given metric is not found.

requests_per_process()
Return the number of requests for the backend per process.

Returns

a list of tuples with 2 elements

1. process number of HAProxy

2. requests

Return type `list`

server(*name*)
Return a Server object

Parameters **name** (*string*) – Name of the server

Returns `Server` object

Return type haproxyadmin.Server

servers(*name=None*)
Return Server object for each server.

Parameters **name** (*string*) – (optional) servername to look for. Defaults to None.

Returns A list of `Server` objects

Return type list

stats_per_process()
Return all stats of the backend per process.

Returns

a list of tuples with 2 elements

1. process number
2. a dict with all stats

Return type `list`

iid
Return the unique proxy ID of the backend.

Note: Because proxy ID is the same across all processes, we return the proxy ID from the 1st process.

Return type `int`

name
Return the name of the backend.

Return type string

process_nb
Return a list of process number in which backend is configured.

Return type list

requests
Return the number of requests.

Return type integer

status
Return the status of the backend.

Return type `string`

Raise `IncosistentData` exception if status is different per process.

3.4 haproxyadmin.server

This module provides the *Server* class which allows to run operation for a server.

class haproxyadmin.server.**Server**(*server_per_proc*, *backendname*)
Build a user-created *Server* for a single server.

Parameters **_server_per_proc** (list) – list of *_Server* objects.

Return type a *Server*.

metric(*name*)
Return the value of a metric.

Performs a calculation on the metric across all HAProxy processes. The type of calculation is either sum or avg and defined in *haproxyadmin.utils.METRICS_SUM* and *haproxyadmin.utils.METRICS_AVG*.

Parameters **name** (any of haproxyadmin.haproxy.SERVER_METRICS) – The name of the metric

Return type number, integer

Raise ValueError when a given metric is not found

requests_per_process()
Return the number of requests for the server per process.

Return type A list of tuple, where 1st element is process number and 2nd element is requests.

setstate(*state*)
Set the state of a server in the backend.

State can be any of the following

- haproxyadmin.STATE_ENABLE: Mark the server UP and checks are re-enabled
- haproxyadmin.STATE_DISABLE: Mark the server DOWN for maintenance and checks disabled.
- haproxyadmin.STATE_READY: Put server in normal mode.
- haproxyadmin.STATE_DRAIN: Remove the server from load balancing.
- haproxyadmin.STATE_MAINT: Remove the server from load balancing and health checks are disabled.

Parameters **state** (string) – state to set.

Returns True if command succeeds otherwise False.

Return type bool

Usage:

```
>>> from haproxyadmin import haproxy, STATE_DISABLE, STATE_ENABLE
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> server = hap.server('member_bkall', backend='backend_proc1')[0]
>>> server.setstate(haproxy.STATE_DISABLE)
True
>>> server.status
'MAINT'
>>> server.setstate(haproxy.STATE_ENABLE)
```

(continues on next page)

(continued from previous page)

```
True
>>> server.status
'no check'
```

setweight(*value*)
Set a weight.

If the value ends with the '%' sign, then the new weight will be relative to the initially configured weight. Absolute weights are permitted between 0 and 256.

Parameters **value** (*integer or string with '%' sign*) – Weight to set

Returns `True` if command succeeds otherwise `False`.

Return type `bool`

Usage:

```
>>> from haproxyadmin import haproxy
>>> hap = haproxy.HAProxy(socket_dir='/run/haproxy')
>>> server = hap.server('member_bkall', backend='backend_proc1')[0]
>>> server.weight
100
>>> server.setweight('20%')
True
>>> server.weight
20
>>> server.setweight(58)
True
>>> server.weight
58
```

shutdown()
Terminate all the sessions attached to the specified server.

Returns `True` if command succeeds otherwise `False`.

Return type `bool`

stats_per_process()
Return all stats of the server per process.

Returns

A list of tuple 2 elements

1. process number
2. a dict with all stats

Return type `list`

address
The assigned address of server.

Getter

rtype `string`

Setter

param address address to set.

type address `string`

rtype bool

check_code
Return the check code.

Return type integer

check_status
Return the check status.

Return type string

last_agent_check
Return the last agent check contents or textual error.

Return type string

last_status
Return the last health check contents or textual error.

Return type string

name
Return the name of the server.

Return type string

port
The assigned port of server.

Getter

rtype string

Setter

param port port to set.

type port string

rtype bool

process_nb
Return a list of process number in which backend server is configured.

Returns a list of process numbers.

Return type list

requests
Return the number of requests.

Return type integer

sid
Return the unique proxy server ID of the server.

Note: Because server ID is the same across all processes, we return the proxy ID from the 1st process.

Return type int

status
Return the status of the server.

Return type string

Raise IncosistentData exception if status is different per process

weight
Return the weight.

Return type integer

Raise IncosistentData exception if weight is different per process

3.5 haproxyadmin.internal.haproxy

This module provides the main class that is used within haproxyadmin for creating object to work with a single HAProxy process. All other internal classes use this class to send commands to HAProxy process.

class haproxyadmin.internal.haproxy.**_HAProxyProcess**(*socket_file*, *retry=3*, *retry_interval=2*)
An object to a single HAProxy process.

It acts as a communication pipe between the caller and individual HAProxy process using UNIX stats socket.

Parameters

- **socket_file** (*string*) – Full path of socket file.
- **retry** (*integer*) – (optional) Number of connect retries (defaults to 3)
- **retry_interval** (*integer*) – (optional) Interval time in seconds between retries (defaults to 2)

backends(*name=None*)
Build _backend objects for each backend.

Parameters **name** (*string*) – (optional) backend name, defaults to None

Returns a list of _backend objects for each backend

Return type list

backends_stats(*iid=-1*)
Build the data structure for backends

If iid is set then builds a structure only for the particul backend.

Parameters **iid** (string) – (optinal) unique proxy id of a backend.

Retur a dictinary with backend information.

Return type dict

command(*command*, *full_output=False*)
Send a command to HAProxy over UNIX stats socket.

Newline character returned from haproxy is stripped off.

Parameters

- **command** (*string*) – A valid command to execute
- **full_output** (bool) – (optional) Return all output, by default returns only the 1st line of the output

Returns 1st line of the output or the whole output as a list

Return type `string` or `list` if full_output is True

frontends(*name=None*)
Build `_Frontend` objects for each frontend.

Parameters **name** (`string`) – (optional) backend name, defaults to `None`

Returns a list of `_Frontend` objects for each backend

Return type `list`

frontends_stats(*iid=-1*)
Build the data structure for frontends

If `iid` is set then builds a structure only for the particular frontend.

Parameters **iid** (`string`) – (optinal) unique proxy id of a frontend.

Retur a dictinary with frontend information.

Return type `dict`

proc_info()
Return a dictionary containing information about HAProxy daemon.

Return type dictionary, see utils.info2dict() for details

stats(*iid=-1*, *obj_type=-1*, *sid=-1*)
Return a nested dictionary containing backend information.

Parameters

- **iid** (`string`) – unique proxy id, applicable for frontends and backends.
- **obj_type** (`integer`) – selects the type of dumpable objects
  - 1 for frontends
  - 2 for backends
  - 4 for servers
  - -1 for everything.

  These values can be ORed, for example:

  1 + 2 = 3 -> frontend + backend. 1 + 2 + 4 = 7 -> frontend + backend + server.
- **sid** (`integer`) – a server ID, -1 to dump everything.

Return type dict, see `utils.stat2dict` for details on the structure

3.6 haproxyadmin.internal.frontend

This module provides a class, which is used within haproxyadmin for creating a object to work with a frontend. This object is associated only with a single HAProxy process.

class haproxyadmin.internal.frontend.**_Frontend**(*hap_process*, *name*, *iid*)
Class for interacting with a frontend in one HAProxy process.

Parameters

- **hap_process** – a `_HAProxyProcess` object.
- **name** (`string`) – frontend name.

- **iid** (integer) – unique proxy id of the frontend.

command(*cmd*)
Run command to HAProxy

Parameters **cmd** (string) – a valid command to execute.

Returns 1st line of the output.

Return type string

metric(*name*)
Return the value of a metric

stats()
Build dictionary for all statistics reported by HAProxy.

return A dictionary with statistics

rtype dict

8. split internal to multiple files

stats_data()
Return stats data

Return type utils.CSVLine object

HAProxy assigns unique ids to each object during the startup. The id can change when configuration changes, objects order is reshuffled or additions/removals take place. In those cases the id we store at the instantiation of the object may reference to another object or even to non-existent object when configuration takes places afterwards.

The technique we use is quite simple. When an object is created we store the name and the id. In order to detect if iid is changed, we simply send a request to fetch data only for the given iid and check if the current id points to an object of the same type (frontend, backend, server) which has the same name.

iid
Return Proxy ID

name
Return a string which is the name of the frontend

3.7 haproxyadmin.internal.backend

This module provides a class, which is used within haproxyadmin for creating a object to work with a backend. This object is associated only with a single HAProxy process.

class haproxyadmin.internal.backend.**_Backend**(*hap_process*, *name*, *iid*)
Class for interacting with a backend in one HAProxy process.

Parameters

- **hap_process** – a :class::*_HAProxyProcess* object.
- **name** (string) – backend name.
- **iid** (integer) – unique proxy id of the backend.

command(*cmd*)
Send command to HAProxy

Parameters **cmd** (string) – command to send

Returns the output of the command

Return type string

servers(*name=None*)
Return a list of _Server objects for each server of the backend.

Parameters **name** (string) – (optional): server name to lookup, defaults to None.

stats()
Build dictionary for all statistics reported by HAProxy.

Returns A dictionary with statistics

Return type dict

stats_data()
Return stats data

Check documentation of stats_data method in _Frontend.

Return type utils.CSVLine object

iid
Return Proxy ID

name
Return a string which is the name of the backend

process_nb
Return the process number of the haproxy process

Return type int

3.8 haproxyadmin.internal.server

This module provides a class, which is used within haproxyadmin for creating a object to work with a server. This object is associated only with a single HAProxy process.

class haproxyadmin.internal.server.**_Server**(*backend*, *name*, *sid*)
Class for interacting with a server of a backend in one HAProxy.

Parameters

- **backend** – a _Backend object in which server is part of.
- **name** (string) – server name.
- **sid** (string) – server id (unique inside a proxy).

stats()
Build dictionary for all statistics reported by HAProxy.

Returns A dictionary with statistics

Return type dict

stats_data()
Return stats data

Check documentation of stats_data method in _Frontend.

Return type utils.CSVLine object

name
Return the name of the backend server.

sid
Return server id

3.9 haproxyadmin.utils

This module provides utility functions and classes that are used within haproxyadmin.

class haproxyadmin.utils.**CSVLine**(*parts*)
An object that holds field/value of a CSV line.

The field name becomes the attribute of the class. Needs the header line of CSV during instantiation.

Parameters **parts** (*list*) – A list with field values

Usage:

```
>>> from haproxyadmin import utils
>>> heads = ['pxname', 'type', 'lbtol']
>>> parts = ['foor', 'backend', '444']
>>> utils.CSVLine.heads = heads
>>> csvobj = utils.CSVLine(parts)
>>> csvobj.pxname
'foor'
>>> csvobj.type
'backend'
>>> csvobj.lbtol
'444'
>>> csvobj.bar
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/.../haproxyadmin/haproxyadmin/utils.py", line 341, in __getattr__
    _index = self.heads.index(attr)
ValueError: 'bar' is not in list
```

haproxyadmin.utils.**calculate**(*name*, *metrics*)
Perform the appropriate calculation across a list of metrics.

Parameters

- **name** (string) – name of the metric.
- **metrics** (list) – a list of metrics. Elements need to be either int or float type number.

Returns either the sum or the average of metrics.

Return type integer

Raise ValueError when matric name has unknown type of calculation.

haproxyadmin.utils.**check_command**(*results*)
Check if command was successfully executed.

After a command is executed. We care about the following cases:

- The same output is returned by all processes
- If output matches to a list of outputs which indicate that command was valid

Parameters **results** (list) – a list of tuples with 2 elements.

1. process number of HAProxy
2. message returned by HAProxy

Returns True if command was successfully executed otherwise False.

Return type bool

Raise *MultipleCommandResults* when output differers.

haproxyadmin.utils.**check_command_addr_port**(*change_type*, *results*)

Check if command to set port or address was successfully executed.

Unfortunately, haproxy returns many different combinations of output when we change the address or the port of the server and trying to determine if address or port was successfully changed isn't that trivial.

So, after we change address or port, we check if the same output is returned by all processes and we also check if a collection of specific strings are part of the output. This is a suboptimal solution, but I couldn't come up with something more elegant.

Parameters

- **change_type** (string) – either addr or port
- **results** (list) – a list of tuples with 2 elements.
  1. process number of HAProxy
  2. message returned by HAProxy

Returns True if command was successfully executed otherwise False.

Return type bool

Raise *MultipleCommandResults*, *CommandFailed* and ValueError.

haproxyadmin.utils.**check_output**(*output*)

Check if output contains any error.

Several commands return output which we need to return back to the caller. But, before we return anything back we want to perform a sanity check on on the output in order to catch wrong input as it is impossible to perform any sanitization on values/patterns which are passed as input to the command.

Parameters **output** (list) – output of the command.

Returns True if no errors found in output otherwise False.

Return type bool

haproxyadmin.utils.**cmd_across_all_procs**(*hap_objects*, *method*, **arg*, ***kargs*)

Return the result of a command executed in all HAProxy process.

Note: Objects must have a property with the name 'process_nb' which returns the HAProxy process number.

Parameters

- **hap_objects** (list) – a list of objects.
- **method** – a valid method for the objects.

Returns

list of 2-item tuple

1. HAProxy process number
2. what the method returned

Return type list

haproxyadmin.utils.**compare_values**(*values*)

Run an intersection test across values returned by processes.

It is possible that not all processes return the same value for certain keys(status, weight etc) due to various reasons. We must detect these cases and either return the value which is the same across all processes or raise <IncosistentData>.

Parameters **values** (list) – a list of tuples with 2 elements.

1. process number of HAProxy process returned the data
2. value returned by HAProxy process.

Returns value

Return type string

Raise *IncosistentData*.

haproxyadmin.utils.**connected_socket**(*path*)

Check if socket file is a valid HAProxy socket file.

We send a 'show info' command to the socket, build a dictionary structure and check if 'Name' key is present in the dictionary to confirm that there is a HAProxy process connected to it.

Parameters **path** (string) – file name path

Returns True is socket file is a valid HAProxy stats socket file False otherwise

Return type bool

haproxyadmin.utils.**converter**(*value*)

Tries to convert input value to an integer.

If input can be safely converted to number it returns an int type. If input is a valid string but not an empty one it returns that. In all other cases we return None, including the ones which an TypeError exception is raised by int(). For floating point numbers, it truncates towards zero.

Why are we doing this? HAProxy may return for a metric either a number or zero or string or an empty string.

It is up to the caller to correctly use the returned value. If the returned value is passed to a function which does math operations the caller has to filtered out possible None values.

Parameters **value** (string) – a value to convert to int.

Return type integer or ``string or None if value can't be converted to int or to string.

Usage:

```
>>> from haproxyadmin import utils
>>> utils.converter('0')
0
>>> utils.converter('13.5')
13
>>> utils.converter('13.5f')
'13.5f'
>>> utils.converter('')
```

(continues on next page)

(continued from previous page)

```
>>> utils.converter(' ')
>>> utils.converter('UP')
'UP'
>>> utils.converter('UP 1/2')
'UP 1/2'
>>>
```

haproxyadmin.utils.**elements_of_list_same**(*iterator*)

Check is all elements of an iterator are equal.

Parameters **iterator** (list) – a iterator

Return type bool

Usage:

```
>>> from haproxyadmin import utils
>>> iterator = ['OK', 'ok']
>>> utils.elements_of_list_same(iterator)
False
>>> iterator = ['OK', 'OK']
>>> utils.elements_of_list_same(iterator)
True
>>> iterator = [22, 22, 22]
>>> utils.elements_of_list_same(iterator)
True
>>> iterator = [22, 22, 222]
>>> utils.elements_of_list_same(iterator)
False
```

haproxyadmin.utils.**info2dict**(*raw_info*)

Build a dictionary structure from the output of 'show info' command.

Parameters **raw_info** (list) – data returned by 'show info' UNIX socket command

Returns A dictionary with the following keys/values(examples)

```
{
    Name: HAProxy
    Version: 1.4.24
    Release_date: 2013/06/17
    Nbproc: 1
    Process_num: 1
    Pid: 1155
    Uptime: 5d 4h42m16s
    Uptime_sec: 448936
    Memmax_MB: 0
    Ulimit-n: 131902
    Maxsock: 131902
    Maxconn: 65536
    Maxpipes: 0
    CurrConns: 1
    PipesUsed: 0
    PipesFree: 0
    Tasks: 819
    Run_queue: 1
    node: node1
    description:
}
```

Return type dict

haproxyadmin.utils.**is_unix_socket**(*path*)
Return True if path is a valid UNIX socket otherwise False.

Parameters **path** (string) – file name path

Return type bool

haproxyadmin.utils.**isint**(*value*)
Check if input can be converted to an integer

Parameters **value** (a string or int) – value to check

Returns True if value can be converted to an integer

Return type bool

Raise ValueError when value can't be converted to an integer

haproxyadmin.utils.**should_die**(*old_implementation*)
Build a decorator to control exceptions.

When a function raises an exception in some cases we don't care for the reason but only if the function run successfully or not. We add an extra argument to the decorated function with the name die to control this behavior. When it is set to True, which is the default value, it raises any exception raised by the decorated function. When it is set to False it returns True if decorated function run successfully or False if an exception was raised.

haproxyadmin.utils.**stat2dict**(*csv_data*)
Build a nested dictionary structure.

Parameters **csv_data** (list) – data returned by 'show stat' command in a CSV format.

Returns

a nested dictionary with all counters/settings found in the input. Following is a sample of the structure:

```
{
    'backends': {
        'acq-misc': {
                            'stats': { _CSVLine object },
                            'servers': {
                                'acqrdb-01': { _CSVLine object },
                                'acqrdb-02': { _CSVLine object },
                                ...
                                }
                        },
        ...
        },
    'frontends': {
        'acq-misc': { _CSVLine object },
        ...
        },
    ...
}
```

Return type dict

3.10 haproxyadmin.exceptions

This module contains the set of haproxyadmin' exceptions with the following hierarchy:

```
HAProxyBaseError
├── CommandFailed
├── HAProxyDataError
│   ├── IncosistentData
│   └── MultipleCommandResults
└── HAProxySocketError
    ├── SocketApplicationError
    ├── SocketConnectionError
    ├── SocketPermissionError
    ├── SocketTimeout
    └── SocketTransportError
```

exception haproxyadmin.exceptions.**CommandFailed**(*message=''*)
Raised when a command to HAProxy returned an error.

exception haproxyadmin.exceptions.**HAProxyBaseError**(*message=''*)
haproxyadmin base exception.

Parameters **message** (string) – error message.

exception haproxyadmin.exceptions.**HAProxyDataError**(*results*)
Base DataError class.

Parameters **results** (list of list) – A structure which contains data returned be each socket.

exception haproxyadmin.exceptions.**HAProxySocketError**(*socket_file*)
Base SocketError class.

Parameters **socket_file** (string) – socket file.

exception haproxyadmin.exceptions.**IncosistentData**(*results*)
Data across all processes is not the same.

exception haproxyadmin.exceptions.**MultipleCommandResults**(*results*)
Command returned different results per HAProxy process.

exception haproxyadmin.exceptions.**SocketApplicationError**(*socket_file*)
Raised when we connect to a socket and HAProxy is not bound to it.

exception haproxyadmin.exceptions.**SocketConnectionError**(*socket_file*)
Raised when socket file is not bound to a process.

exception haproxyadmin.exceptions.**SocketPermissionError**(*socket_file*)
Raised when permissions are not granted to access socket file.

exception haproxyadmin.exceptions.**SocketTimeout**(*socket_file*)
Raised when we timeout on the socket.

exception haproxyadmin.exceptions.**SocketTransportError**(*socket_file*)
Raised when endpoint of socket hasn't closed an old connection.

Note: It only occurs in cases where HAProxy is ~90% CPU utilization for processing traffic and we reconnect to the socket too fast and as a result HAProxy doesn't have enough time to close the previous connection.

3.11 Constants

3.11.1 Metric names

Various stats field names for which a value can be retrieved by using `metric` method available in all public and internal interfaces.

haproxyadmin.**FRONTEND_METRICS = a list of metric names for retrieving varius statistics fo**

haproxyadmin.**BACKEND_METRICS = a list of metric names for retrieving varius statistics for**

haproxyadmin.**SERVER_METRICS = a list of metric names for retrieving varius statistics for s**

3.11.2 Aggregation rules

The following 2 constants define the type of aggregation, either sum or average, which is performed for values returned by all HAProxy processes.

haproxyadmin.utils.**METRICS_SUM = ['CompressBpsIn', 'CompressBpsOut', 'CompressBpsRateLim',**
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

haproxyadmin.utils.**METRICS_AVG = ['act', 'bck', 'check_duration', 'ctime', 'downtime', 'las**
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

3.11.3 Valid server states

A list of constants to use in `setstate` of *Server* to change the state of a server.

CHAPTER 4

CHANGES

- Split internal classes to individual modules
- Try to smarter when we return address/port
- Remove useless object inheritance
- setup.cfg: We don't need psutil anymore
- DOC: Update docstrings

4.1 0.2.2

- RELEASE 0.2.2 version
- Revert "Add 'slim' metric for servers"
- DOC: One more try to get this right
- DOC: Update docstring for address method
- Add support for changing address and port of a server
- DOC: Fix various documentation issues
- Add 'slim' metric for servers
- Don't check if ACL/MAP is a file
- Return empty list if a acl doesn't have any entries
- DOC: Change python version we use for development
- Ignore Python 3 class hierarchy of OSError errors
- Return empty list if a map doesn't have any entries
- Fix example code for show_map function
- Fix incorrect module path for constants in docstring

- Add support for slim metric to Server object
- List server metric names in alphabetic order
- Added setaddress and address to Servers
- Fix docstrings
- Remove unused variables
- Ignore Python 3 class hierarchy of OSError errors
- Update installation instructions

4.2 0.2.1

- RELEASE 0.2.1 version
- Reorder inclusion of modules
- Add docstring for isint()
- Simplify conditional statement
- Fix typos in a docstring
- Reorder inclusion of modules and remove unused exceptions
- Return False when a file isn't a valid stats socket
- Update copyright
- Pass keyword parameters in format method, fix #1

4.3 0.2.0

- RELEASE 0.2.0 version
- Refactor constants for metrics
- Include a module docstring

4.4 0.1.12

- RELEASE 0.1.12 version
- Return zero rather None for metrics without value

4.5 0.1.11

- RELEASE 0.1.11 version
- Make sure we clear out possible previous errors
- Remove unnecessary keyword argument

4.6 0.1.10

- RELEASE 0.1.10 version
- Implement a proper retry logic for socket failures

4.7 0.1.9

- RELEASE 0.1.9 version
- Improve the way we internally use values for metrics

4.8 0.1.8

- RELEASE 0.1.8 version
- Remove unnecessary filtering of empty values

CHAPTER 5

TODO

1. Add support for enabling/disabling health/agent checks
2. TLS ticket operations
3. Add support for TLS ticket operations
4. Add support for OCSP stapling
5. Add support for changing server's IP
6. Add support for DNS resolvers
7. Add support for dumping sessions
8. make internal._HAProxyProcess.send_command() to return file type object as it will avoid to run through the list 2 times.
9. Test against hapee, haproxy-1.6dev4
10. Investigate the use of __slots__ in utils.CSVLine as it could speed up the library when we create 100K objects

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

Symbols

A

B

C

D

E

R

S

T

U

V

W