HTTPretty Documentation

Release 1.1.3

Gabriel Falcao

May 24, 2021

CONTENTS

HTTP Client mocking tool for Python created by Gabriel Falcão . It provides a full fake TCP socket module. Inspired by FakeWeb

Looking for the Github Repository ?

Python Support:

- **3.6**
- **3.7**
- **3.8**
- **3.9**

Github

CHAPTER

ONE

WHAT IS HTTPRETTY ?

Once upon a time a python developer wanted to use a RESTful api, everything was fine but until the day he needed to test the code that hits the RESTful API: what if the API server is down? What if its content has changed ?

Don't worry, HTTPretty is here for you:

```
import logging
import requests
import httpretty

from sure import expect

logging.getLogger('httpretty.core').setLevel(logging.DEBUG)

@httpretty.activate(allow_net_connect=False)
def test_yipit_api_returning_deals():
    httpretty.register_uri(httpretty.GET, "http://api.yipit.com/v1/deals/",
                           body='[{"title": "Test Deal"}]',
                           content_type="application/json")

    response = requests.get('http://api.yipit.com/v1/deals/')

    expect(response.json()).to.equal([{"title": "Test Deal"}])
```

1.1 A more technical description

HTTPretty is a python library that swaps the modules `socket` and `ssl` with fake implementations that intercept HTTP requests at the level of a TCP connection.

It is inspired on Ruby's FakeWeb.

If you come from the Ruby programming language this would probably sound familiar :smiley:

1.2 Installing

Installing httpretty is as easy as:

```
pip install httpretty
```

CHAPTER

TWO

DEMO

2.1 expecting a simple response body

```
import requests
import httpretty

def test_one():
    httpretty.enable(verbose=True, allow_net_connect=False)  # enable HTTPretty so that␣
↪it will monkey patch the socket module
    httpretty.register_uri(httpretty.GET, "http://yipit.com/",
                           body="Find the best daily deals")

    response = requests.get('http://yipit.com')

    assert response.text == "Find the best daily deals"

    httpretty.disable()  # disable afterwards, so that you will have no problems in code␣
↪that uses that socket module
    httpretty.reset()    # reset HTTPretty state (clean up registered urls and request␣
↪history)
```

2.2 making assertions in a callback that generates the response body

```
import requests
import json
import httpretty

@httpretty.activate
def test_with_callback_response():
  def request_callback(request, uri, response_headers):
      content_type = request.headers.get('Content-Type')
      assert request.body == '{"nothing": "here"}', 'unexpected body: {}'.format(request.
↪body)
      assert content_type == 'application/json', 'expected application/json but received␣
↪Content-Type: {}'.format(content_type)
      return [200, response_headers, json.dumps({"hello": "world"})]

  httpretty.register_uri(
```

(continues on next page)

(continued from previous page)

```
    httpretty.POST, "https://httpretty.example.com/api",
    body=request_callback)

response = requests.post('https://httpretty.example.com/api', headers={'Content-Type':
↪'application/json'}, data='{"nothing": "here"}')

expect(response.json()).to.equal({"hello": "world"})
```

2.3 Link headers

Tests link headers by using the *adding_headers* parameter.

```
import requests
from sure import expect
import httpretty


@httpretty.activate
def test_link_response():
    first_url = "http://foo-api.com/data"
    second_url = "http://foo-api.com/data?page=2"
    link_str = "<%s>; rel='next'" % second_url

    httpretty.register_uri(
        httpretty.GET,
        first_url,
        body='{"success": true}',
        status=200,
        content_type="text/json",
        adding_headers={"Link": link_str},
    )
    httpretty.register_uri(
        httpretty.GET,
        second_url,
        body='{"success": false}',
        status=500,
        content_type="text/json",
    )
    # Performs a request to `first_url` followed by some testing
    response = requests.get(first_url)
    expect(response.json()).to.equal({"success": True})
    expect(response.status_code).to.equal(200)
    next_url = response.links["next"]["url"]
    expect(next_url).to.equal(second_url)

    # Follow the next URL and perform some testing.
    response2 = requests.get(next_url)
    expect(response2.json()).to.equal({"success": False})
    expect(response2.status_code).to.equal(500)
```

CHAPTER

THREE

MOTIVATION

When building systems that access external resources such as RESTful webservices, XMLRPC or even simple HTTP requests, we stumble in the problem:

> *"I'm gonna need to mock all those requests"*

It can be a bit of a hassle to use something like `mock.Mock` to stub the requests, this can work well for low-level unit tests but when writing functional or integration tests we should be able to allow the http calls to go through the TCP socket module.

HTTPretty monkey patches Python's `socket` core module with a fake version of the module.

Because HTTPretty implements a fake the modules `socket` and `ssl` you can use write tests to code against any HTTP library that use those modules.

CHAPTER

FOUR

GUIDES

A series of guides to using HTTPretty for various interesting purposes.

4.1 Matching URLs via regular expressions

You can pass a compiled regular expression via `re.compile()`, for example for intercepting all requests to a specific host.

Example:

```
import re
import requests
import httpretty


@httpretty.activate(allow_net_connect=False, verbose=True)
def test_regex():
    httpretty.register_uri(httpretty.GET, re.compile(r'.*'), status=418)

    response1 = requests.get('http://foo.com')
    assert response1.status_code == 418

    response2 = requests.get('http://test.com')
    assert response2.status_code == 418
```

4.2 Response Callbacks

You can use the *body* parameter of *`register_uri()`* in useful, practical ways because it accepts a `callable()` as value.

As matter of example, this is analogous to defining routes in Flask when combined with *matching urls via regular expressions*

This analogy breaks down, though, because HTTPretty does not provide tools to make it easy to handle cookies, parse querystrings etc.

So far this has been a deliberate decision to keep HTTPretty operating mostly at the TCP socket level.

Nothing prevents you from being creative with callbacks though, and as you will see in the examples below, the request parameter is an instance of *`HTTPrettyRequest`* which has everything you need to create elaborate fake APIs.

4.2.1 Defining callbacks

The body parameter callback must:

- Accept 3 arguments:
  - *request* - *HTTPrettyRequest*
  - *uri* - `str`
  - *headers* - `dict` with default response headers (including the ones from the parameters `adding_headers` and `forcing_headers` of *register_uri()*
- Return 3 a tuple (or list) with 3 values
  - `int` - HTTP Status Code
  - `dict` - Response Headers
  - `st` - Response Body

Important: The **Content-Length** should match the byte length of the body.

Changing **Content-Length** it in your handler can cause your HTTP client to misbehave, be very intentional when modifying it in our callback.

The suggested way to manipulate headers is by modifying the response headers passed as argument and returning them in the tuple at the end.

```
  from typing import Tuple
  from httpretty.core import HTTPrettyRequest

  def my_callback(
          request: HTTPrettyRequest,
          url: str,
          headers: dict

      ) -> Tuple[int, dict, str]:

      headers['Content-Type'] = 'text/plain'
      return (200, headers, "the body")

HTTPretty.register_uri(HTTPretty.GET, "https://test.com", body=my_callback)
```

4.2.2 Debug requests interactively with ipdb

The library ipdb comes in handy to introspect the request interactively with auto-complete via IPython.

```
import re
import json
import requests
from httpretty import httprettified, HTTPretty


@httprettified(verbose=True, allow_net_connect=False)
def test_basic_body():
```

(continues on next page)

(continued from previous page)

```
    def my_callback(request, url, headers):
        body = {}
        import ipdb;ipdb.set_trace()
        return (200, headers, json.dumps(body))

    # Match any url via the regular expression
    HTTPretty.register_uri(HTTPretty.GET, re.compile(r'.*'), body=my_callback)
    HTTPretty.register_uri(HTTPretty.POST, re.compile(r'.*'), body=my_callback)

    # will trigger ipdb
    response = requests.post('https://test.com', data=json.dumps({'hello': 'world'}))
```

4.2.3 Emulating timeouts

In the bug report #430 the contributor @mariojonke provided a neat example of how to emulate read timeout errors by "waiting" inside of a body callback.

```
import requests, time
from threading import Event

from httpretty import httprettified
from httpretty import HTTPretty


@httprettified(allow_net_connect=False)
def test_read_timeout():
    event = Event()
    wait_seconds = 10
    connect_timeout = 0.1
    read_timeout = 0.1

    def my_callback(request, url, headers):
        event.wait(wait_seconds)
        return 200, headers, "Received"

    HTTPretty.register_uri(
        HTTPretty.GET, "http://example.com",
        body=my_callback
    )

    requested_at = time.time()
    try:
        requests.get(
            "http://example.com",
            timeout=(connect_timeout, read_timeout))
    except requests.exceptions.ReadTimeout:
        pass

    event_set_at = time.time()
    event.set()
```

(continues on next page)

(continued from previous page)

```
    now = time.time()

    assert now - event_set_at < 0.2
    total_duration = now - requested_at
    assert total_duration < 0.2
```

CHAPTER

FIVE

ACKNOWLEDGEMENTS

5.1 Caveats

5.1.1 `forcing_headers` + `Content-Length`

When using the `forcing_headers` option make sure to add the header `Content-Length` otherwise calls using `requests` will try to load the response endlessly.

5.1.2 Supported Libraries

Because HTTPretty works in the socket level it should work with any HTTP client libraries, although it is battle tested against:

- requests
- httplib2
- urllib2

CHAPTER

SIX

API REFERENCE

6.1 register_uri

classmethod httpretty.**register_uri**(*method*, *uri*, *body='{"message": "HTTPretty :)"}'*, *adding_headers=None*, *forcing_headers=None*, *status=200*, *responses=None*, *match_querystring=False*, *priority=0*, ***headers*)

```
import httpretty


def request_callback(request, uri, response_headers):
    content_type = request.headers.get('Content-Type')
    assert request.body == '{"nothing": "here"}', 'unexpected body: {}'.
↪format(request.body)
    assert content_type == 'application/json', 'expected application/json but␣
↪received Content-Type: {}'.format(content_type)
    return [200, response_headers, json.dumps({"hello": "world"})]

httpretty.register_uri(
    HTTPretty.POST, "https://httpretty.example.com/api",
    body=request_callback)


with httpretty.enabled():
    requests.post('https://httpretty.example.com/api', data='{"nothing": "here"}',␣
↪headers={'Content-Type': 'application/json'})

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
```

Parameters

- **method** – one of `httpretty.GET`, `httpretty.PUT`, `httpretty.POST`, `httpretty.DELETE`, `httpretty.HEAD`, `httpretty.PATCH`, `httpretty.OPTIONS`, `httpretty.CONNECT`
- **uri** – a string or regex pattern (e.g.: **"https://httpbin.org/ip"**)
- **body** – a string, defaults to `{"message": "HTTPretty :)"}`
- **adding_headers** – dict - headers to be added to the response
- **forcing_headers** – dict - headers to be forcefully set in the response

- **status** – an integer, defaults to **200**
- **responses** – a list of entries, ideally each created with *Response()*
- **priority** – an integer, useful for setting higher priority over previously registered urls. defaults to zero
- **match_querystring** – bool - whether to take the querystring into account when matching an URL
- **headers** – headers to be added to the response

Warning: When using a port in the request, add a trailing slash if no path is provided otherwise Httpretty will not catch the request. Ex: `httpretty.register_uri(httpretty.GET, 'http://fakeuri.com:8080/', body='{"hello":"world"}')`

6.2 enable

classmethod httpretty.**enable**(*allow_net_connect=True*, *verbose=False*)

Enables HTTPretty.

Parameters

- **allow_net_connect** – boolean to determine if unmatched requests are forwarded to a real network connection OR throw *httpretty.errors.UnmockedError*.
- **verbose** – boolean to set HTTPretty's logging level to DEBUG

```
import re, json
import httpretty

httpretty.enable(allow_net_connect=True, verbose=True)

httpretty.register_uri(
    httpretty.GET,
    re.compile(r'http://.*'),
    body=json.dumps({'man': 'in', 'the': 'middle'})
)

response = requests.get('https://foo.bar/foo/bar')

response.json().should.equal({
    "man": "in",
    "the": "middle",
})
```

Warning: after calling this method the original `socket` is replaced with *httpretty.core.fakesock*. Make sure to call *disable()* after done with your tests or use the `httpretty.enabled` as decorator or context-manager

6.3 disable

classmethod httpretty.**disable**()

Disables HTTPretty entirely, putting the original `socket` module back in its place.

```
import re, json
import httpretty

httpretty.enable()
# request passes through fake socket
response = requests.get('https://httpbin.org')

httpretty.disable()
# request uses real python socket module
response = requests.get('https://httpbin.org')
```

Note: This method does not call `httpretty.core.reset()` automatically.

6.4 is_enabled

classmethod httpretty.**is_enabled**()

Check if HTTPretty is enabled

Returns bool

```
import httpretty

httpretty.enable()
assert httpretty.is_enabled() == True

httpretty.disable()
assert httpretty.is_enabled() == False
```

6.5 last_request

httpretty.**last_request**()

Returns the last *HTTPrettyRequest*

6.6 latest_requests

httpretty.**latest_requests**()
returns the history of made requests

6.7 activate

httpretty.**activate**
alias of *httpretty.core.httprettified*

6.8 httprettified

httpretty.core.**httprettified**(*test=None*, *allow_net_connect=True*, *verbose=False*)
decorator for test functions

Tip: Also available under the alias `httpretty.activate()`

Parameters **test** – a callable

example usage with nosetests

```
import sure
from httpretty import httprettified

@httprettified
def test_using_nosetests():
    httpretty.register_uri(
        httpretty.GET,
        'https://httpbin.org/ip'
    )

    response = requests.get('https://httpbin.org/ip')

    response.json().should.equal({
        "message": "HTTPretty :)"
    })
```

example usage with unittest module

```
import unittest
from sure import expect
from httpretty import httprettified

@httprettified
class TestWithPyUnit(unittest.TestCase):
    def test_httpbin(self):
        httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip')
        response = requests.get('https://httpbin.org/ip')
```

(continues on next page)

(continued from previous page)

```
        expect(response.json()).to.equal({
            "message": "HTTPretty :)"
        })
```

6.9 enabled

httpretty.**enabled**
alias of *httpretty.core.httprettized*

6.10 httprettized

class httpretty.core.**httprettized**(*allow_net_connect=True*, *verbose=False*)
context-manager for enabling HTTPretty.

Tip: Also available under the alias `httpretty.enabled()`

```
import json
import httpretty

httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip', body=json.dumps({
↪'origin': '42.42.42.42'}))
with httpretty.enabled():
    response = requests.get('https://httpbin.org/ip')

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
assert response.json() == {'origin': '42.42.42.42'}
```

6.11 HTTPrettyRequest

class httpretty.core.**HTTPrettyRequest**(*headers*, *body=''*, *sock=None*, *path_encoding='iso-8859-1'*)
Represents a HTTP request. It takes a valid multi-line, `\r\n` separated string with HTTP headers and parse them out using the internal *parse_request* method.

It also replaces the *rfile* and *wfile* attributes with `io.BytesIO` instances so that we guarantee that it won't make any I/O, neither for writing nor reading.

It has some convenience attributes:

`headers` -> a mimetype object that can be cast into a dictionary, contains all the request headers

`protocol` -> the protocol of this host, inferred from the port of the underlying fake TCP socket.

`host` -> the hostname of this request.

`url` -> the full url of this request.

`path` -> the path of the request.

`method` -> the HTTP method used in this request.

querystring -> a dictionary containing lists with the attributes. Please notice that if you need a single value from a query string you will need to get it manually like:

body -> the request body as a string.

parsed_body -> the request body parsed by parse_request_body.

```
>>> request.querystring
{'name': ['Gabriel Falcao']}
>>> print request.querystring['name'][0]
```

property method
the HTTP method used in this request

parse_querystring(*qs*)
parses an UTF-8 encoded query string into a dict of string lists

Parameters **qs** – a querystring

Returns a dict of lists

parse_request_body(*body*)
Attempt to parse the post based on the content-type passed. Return the regular body if not

Parameters **body** – string

Returns a python object such as dict or list in case the deserialization suceeded. Else returns the given param body

property protocol
the protocol used in this request

querystring
a dictionary containing parsed request body or None if HTTPrettyRequest doesn't know how to parse it. It currently supports parsing body data that was sent under the content`-type` headers values: ``application/json or application/x-www-form-urlencoded

property url
the full url of this recorded request

6.12 HTTPrettyRequestEmpty

class httpretty.core.**HTTPrettyRequestEmpty**
Represents an empty *HTTPrettyRequest* where all its properties are somehow empty or None

6.13 FakeSockFile

class httpretty.core.**FakeSockFile**
Fake socket file descriptor. Under the hood all data is written in a temporary file, giving it a real file descriptor number.

6.14 FakeSSLSocket

class httpretty.core.**FakeSSLSocket**(*sock*, **args*, ***kw*)

Shorthand for *fakesock*

6.15 URIInfo

class httpretty.**URIInfo**(*username=''*, *password=''*, *hostname=''*, *port=80*, *path='/'*, *query=''*, *fragment=''*, *scheme=''*, *last_request=None*)

Internal representation of URIs

Tip: all arguments are optional

Parameters

- **username** –
- **password** –
- **hostname** –
- **port** –
- **path** –
- **query** –
- **fragment** –
- **scheme** –
- **last_request** –

classmethod **from_uri**(*uri*, *entry*)

Parameters

- **uri** – string
- **entry** – an instance of *Entry*

full_url(*use_querystring=True*)

Parameters **use_querystring** – bool

Returns a string with the full url with the format `{scheme}://{credentials}{domain}{path}{query}`

get_full_domain()

Returns a string in the form `{domain}:{port}` or just the domain if the port is 80 or 443

6.16 URIMatcher

class httpretty.**URIMatcher**(*uri*, *entries*, *match_querystring=False*, *priority=0*)

get_next_entry(*method*, *info*, *request*)
Cycle through available responses, but only once. Any subsequent requests will receive the last response

6.17 Entry

class httpretty.**Entry**(*method*, *uri*, *body*, *adding_headers=None*, *forcing_headers=None*, *status=200*, *streaming=False*, ***headers*)
Created by *register_uri()* and stored in memory as internal representation of a HTTP request/response definition.

Parameters

- **method** (*str*) – One of httpretty.GET, httpretty.PUT, httpretty.POST, httpretty.DELETE, httpretty.HEAD, httpretty.PATCH, httpretty.OPTIONS, httpretty.CONNECT.
- **uri** (*str|re.Pattern*) – The URL to match
- **adding_headers** (*dict*) – Extra headers to be added to the response
- **forcing_headers** (*dict*) – Overwrite response headers.
- **status** (*int*) – The status code for the response, defaults to 200.
- **streaming** (*bool*) – Whether should stream the response into chunks via generator.
- **headers** – Headers to inject in the faked response.

Returns containing the request-matching metadata.

Return type httpretty.Entry

> **Warning:** When using the forcing_headers option make sure to add the header Content-Length to match at most the total body length, otherwise some HTTP clients can hang indefinitely.

fill_filekind(*fk*)
writes HTTP Response data to a file descriptor

Parm fk a file-like object

> **Warning:** **side-effect:** this method moves the cursor of the given file object to zero

normalize_headers(*headers*)
Normalize keys in header names so that COntent-tyPe becomes content-type

Parameters **headers** – dict

Returns dict

validate()
validates the body size with the value of the Content-Length header

CHAPTER

SEVEN

MODULES

7.1 Core

class httpretty.core.**EmptyRequestHeaders**
A dict subclass used as internal representation of empty request headers

class httpretty.core.**Entry**(*method*, *uri*, *body*, *adding_headers=None*, *forcing_headers=None*, *status=200*, *streaming=False*, ***headers*)
Created by *register_uri()* and stored in memory as internal representation of a HTTP request/response definition.

Parameters

- **method** (*str*) – One of `httpretty.GET`, `httpretty.PUT`, `httpretty.POST`, `httpretty.DELETE`, `httpretty.HEAD`, `httpretty.PATCH`, `httpretty.OPTIONS`, `httpretty.CONNECT`.
- **uri** (*str|re.Pattern*) – The URL to match
- **adding_headers** (*dict*) – Extra headers to be added to the response
- **forcing_headers** (*dict*) – Overwrite response headers.
- **status** (*int*) – The status code for the response, defaults to `200`.
- **streaming** (*bool*) – Whether should stream the response into chunks via generator.
- **headers** – Headers to inject in the faked response.

Returns containing the request-matching metadata.

Return type httpretty.Entry

> **Warning:** When using the `forcing_headers` option make sure to add the header `Content-Length` to match at most the total body length, otherwise some HTTP clients can hang indefinitely.

fill_filekind(*fk*)
writes HTTP Response data to a file descriptor

Parm fk a file-like object

> **Warning:** **side-effect:** this method moves the cursor of the given file object to zero

normalize_headers(*headers*)
Normalize keys in header names so that `COntent-tyPe` becomes `content-type`

Parameters **headers** – dict

Returns dict

validate()
validates the body size with the value of the `Content-Length` header

class `httpretty.core.`**FakeSSLSocket**(*sock*, **args*, ***kw*)
Shorthand for *fakesock*

class `httpretty.core.`**FakeSockFile**
Fake socket file descriptor. Under the hood all data is written in a temporary file, giving it a real file descriptor number.

class `httpretty.core.`**HTTPrettyRequest**(*headers*, *body=''*, *sock=None*, *path_encoding='iso-8859-1'*)
Represents a HTTP request. It takes a valid multi-line, `\r\n` separated string with HTTP headers and parse them out using the internal *parse_request* method.

It also replaces the *rfile* and *wfile* attributes with `io.BytesIO` instances so that we guarantee that it won't make any I/O, neither for writing nor reading.

It has some convenience attributes:

`headers` -> a mimetype object that can be cast into a dictionary, contains all the request headers

`protocol` -> the protocol of this host, inferred from the port of the underlying fake TCP socket.

`host` -> the hostname of this request.

`url` -> the full url of this request.

`path` -> the path of the request.

`method` -> the HTTP method used in this request.

`querystring` -> a dictionary containing lists with the attributes. Please notice that if you need a single value from a query string you will need to get it manually like:

`body` -> the request body as a string.

`parsed_body` -> the request body parsed by `parse_request_body`.

```
>>> request.querystring
{'name': ['Gabriel Falcao']}
>>> print request.querystring['name'][0]
```

property method
the HTTP method used in this request

parse_querystring(*qs*)
parses an UTF-8 encoded query string into a dict of string lists

Parameters **qs** – a querystring

Returns a dict of lists

parse_request_body(*body*)
Attempt to parse the post based on the content-type passed. Return the regular body if not

Parameters **body** – string

Returns a python object such as dict or list in case the deserialization suceeded. Else returns the given param `body`

property protocol
the protocol used in this request

querystring
a dictionary containing parsed request body or None if HTTPrettyRequest doesn't know how to parse it. It currently supports parsing body data that was sent under the `content-type` headers values: ``application/json` or `application/x-www-form-urlencoded`

property url
the full url of this recorded request

class httpretty.core.**HTTPrettyRequestEmpty**
Represents an empty *HTTPrettyRequest* where all its properties are somehow empty or None

class httpretty.core.**URIInfo**(*username=''*, *password=''*, *hostname=''*, *port=80*, *path='/'*, *query=''*, *fragment=''*, *scheme=''*, *last_request=None*)
Internal representation of URIs

Tip: all arguments are optional

Parameters

- **username** –
- **password** –
- **hostname** –
- **port** –
- **path** –
- **query** –
- **fragment** –
- **scheme** –
- **last_request** –

classmethod from_uri(*uri*, *entry*)

Parameters

- **uri** – string
- **entry** – an instance of *Entry*

full_url(*use_querystring=True*)

Parameters **use_querystring** – bool

Returns a string with the full url with the format `{scheme}://{credentials}{domain}{path}{query}`

get_full_domain()

Returns a string in the form `{domain}:{port}` or just the domain if the port is 80 or 443

httpretty.core.**create_fake_connection**(*address*, *timeout=<object object>*, *source_address=None*)
drop-in replacement for socket.create_connection()

httpretty.core.**fake_getaddrinfo**(*host*, *port*, *family=None*, *socktype=None*, *proto=None*, *flags=None*)
drop-in replacement for socket.getaddrinfo()

httpretty.core.**fake_gethostbyname**(*host*)
drop-in replacement for socket.gethostbyname()

httpretty.core.**fake_gethostname**()
drop-in replacement for socket.gethostname()

httpretty.core.**fake_wrap_socket**(*orig_wrap_socket_fn*, **args*, ***kw*)
drop-in replacement for py:func:*ssl.wrap_socket*

class httpretty.core.**fakesock**
fake socket

class socket(*family=<AddressFamily.AF_INET: 2>*, *type=<SocketKind.SOCK_STREAM: 1>*, *proto=0*, *fileno=None*)
drop-in replacement for socket.socket

bind(*address*)

bind_truesock(*address*)

close()

connect(*address*)

connect_truesock(*request=None*, *address=None*)

create_socket(*address=None*)

fileno()

forward_and_trace(*function_name*, **a*, ***kw*)

getpeercert(**a*, ***kw*)

property host

makefile(*mode='r'*, *bufsize=- 1*)
Returns this fake socket's own tempfile buffer.

If there is an entry associated with the socket, the file descriptor gets filled in with the entry data before being returned.

real_sendall(*data*, **args*, ***kw*)
Sends data to the remote server. This method is called when HTTPretty identifies that someone is trying to send non-http data.

The received bytes are written in this socket's tempfile buffer so that HTTPretty can return it accordingly when necessary.

real_socket_is_connected()

recv(*buffersize=0*, **args*, ***kwargs*)

recv_into(**args*, ***kwargs*)

recvfrom(**args*, ***kwargs*)

recvfrom_into(**args*, ***kwargs*)

send(*data*, **args*, ***kwargs*)

sendall(*data*, **args*, ***kw*)

sendto(**args*, ***kwargs*)

setsockopt(*level*, *optname*, *value*)

settimeout(*new_timeout*)

ssl(*sock*, **args*, ***kw*)

`httpretty.core.`**get_default_thread_timeout**()
sets the default thread timeout for HTTPretty threads

Returns int

`httpretty.core.`**httprettified**(*test=None*, *allow_net_connect=True*, *verbose=False*)
decorator for test functions

Tip: Also available under the alias `httpretty.activate()`

Parameters **test** – a callable

example usage with nosetests

```
import sure
from httpretty import httprettified

@httprettified
def test_using_nosetests():
    httpretty.register_uri(
        httpretty.GET,
        'https://httpbin.org/ip'
    )

    response = requests.get('https://httpbin.org/ip')

    response.json().should.equal({
        "message": "HTTPretty :)"
    })
```

example usage with unittest module

```
import unittest
from sure import expect
from httpretty import httprettified

@httprettified
class TestWithPyUnit(unittest.TestCase):
    def test_httpbin(self):
        httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip')
        response = requests.get('https://httpbin.org/ip')
        expect(response.json()).to.equal({
            "message": "HTTPretty :)"
        })
```

class httpretty.core.**httprettized**(*allow_net_connect=True*, *verbose=False*)

context-manager for enabling HTTPretty.

Tip: Also available under the alias `httpretty.enabled()`

```
import json
import httpretty

httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip', body=json.dumps({
↪'origin': '42.42.42.42'}))
with httpretty.enabled():
    response = requests.get('https://httpbin.org/ip')

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
assert response.json() == {'origin': '42.42.42.42'}
```

class httpretty.core.**httpretty**

manages HTTPretty's internal request/response registry and request matching.

classmethod **Response**(*body*, *method=None*, *uri=None*, *adding_headers=None*, *forcing_headers=None*, *status=200*, *streaming=False*, ***kw*)

Shortcut to create an *Entry* that takes the body as first positional argument.

See also:

the parameters of this function match those of the *Entry* constructor.

Parameters

- **body** (*str*) – The body to return as response..
- **method** (*str*) – One of `httpretty.GET`, `httpretty.PUT`, `httpretty.POST`, `httpretty.DELETE`, `httpretty.HEAD`, `httpretty.PATCH`, `httpretty.OPTIONS`, `httpretty.CONNECT`.
- **uri** (*str|re.Pattern*) – The URL to match
- **adding_headers** (*dict*) – Extra headers to be added to the response
- **forcing_headers** (*dict*) – Overwrite **any** response headers, even "Content-Length".
- **status** (*int*) – The status code for the response, defaults to `200`.
- **streaming** (*bool*) – Whether should stream the response into chunks via generator.
- **kwargs** – Keyword-arguments are forwarded to *Entry*

Returns containing the request-matching metadata.

Return type httpretty.Entry

classmethod **disable**()

Disables HTTPretty entirely, putting the original `socket` module back in its place.

```
import re, json
import httpretty

httpretty.enable()
```

(continues on next page)

(continued from previous page)

```
# request passes through fake socket
response = requests.get('https://httpbin.org')

httpretty.disable()
# request uses real python socket module
response = requests.get('https://httpbin.org')
```

Note: This method does not call `httpretty.core.reset()` automatically.

classmethod enable(*allow_net_connect=True*, *verbose=False*)

Enables HTTPretty.

Parameters

- **allow_net_connect** – boolean to determine if unmatched requests are forwarded to a real network connection OR throw *httpretty.errors.UnmockedError*.
- **verbose** – boolean to set HTTPretty's logging level to DEBUG

```
import re, json
import httpretty

httpretty.enable(allow_net_connect=True, verbose=True)

httpretty.register_uri(
    httpretty.GET,
    re.compile(r'http://.*'),
    body=json.dumps({'man': 'in', 'the': 'middle'})
)

response = requests.get('https://foo.bar/foo/bar')

response.json().should.equal({
    "man": "in",
    "the": "middle",
})
```

Warning: after calling this method the original `socket` is replaced with *httpretty.core.fakesock*. Make sure to call *disable()* after done with your tests or use the `httpretty.enabled` as decorator or context-manager

classmethod historify_request(*headers*, *body=''*, *sock=None*)

appends request to a list for later retrieval

```
import httpretty

httpretty.register_uri(httpretty.GET, 'https://httpbin.org/ip', body='')
with httpretty.enabled():
    requests.get('https://httpbin.org/ip')

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
```

classmethod is_enabled()
Check if HTTPretty is enabled

Returns bool

```
import httpretty

httpretty.enable()
assert httpretty.is_enabled() == True

httpretty.disable()
assert httpretty.is_enabled() == False
```

classmethod match_http_address(*hostname*, *port*)

Parameters

- **hostname** – a string
- **port** – an integer

Returns an `URLMatcher` or `None`

classmethod match_https_hostname(*hostname*)

Parameters **hostname** – a string

Returns an `URLMatcher` or `None`

classmethod match_uriinfo(*info*)

Parameters **info** – an *URIInfo*

Returns a 2-item tuple: (`URLMatcher`, *URIInfo*) or `(None, [])`

classmethod playback(*filename*, *allow_net_connect=True*, *verbose=False*)

```
import io
import json
import requests
import httpretty

with httpretty.record('/tmp/ip.json'):
    data = requests.get('https://httpbin.org/ip').json()

with io.open('/tmp/ip.json') as fd:
    assert data == json.load(fd)
```

Parameters **filename** – a string

Returns

a context-manager

classmethod record(*filename*, *indentation=4*, *encoding='utf-8'*, *verbose=False*, *allow_net_connect=True*, *pool_manager_params=None*)

```
import io
import json
import requests
import httpretty

with httpretty.record('/tmp/ip.json'):
    data = requests.get('https://httpbin.org/ip').json()

with io.open('/tmp/ip.json') as fd:
    assert data == json.load(fd)
```

Parameters

- **filename** – a string
- **indentation** – an integer, defaults to **4**
- **encoding** – a string, defaults to **"utf-8"**

Returns

a context-manager

classmethod register_uri(*method*, *uri*, *body='{"message": "HTTPretty :)"}'*, *adding_headers=None*, *forcing_headers=None*, *status=200*, *responses=None*, *match_querystring=False*, *priority=0*, ***headers*)

```
import httpretty


def request_callback(request, uri, response_headers):
    content_type = request.headers.get('Content-Type')
    assert request.body == '{"nothing": "here"}', 'unexpected body: {}'.
↪format(request.body)
    assert content_type == 'application/json', 'expected application/json but␣
↪received Content-Type: {}'.format(content_type)
    return [200, response_headers, json.dumps({"hello": "world"})]

httpretty.register_uri(
    HTTPretty.POST, "https://httpretty.example.com/api",
    body=request_callback)


with httpretty.enabled():
    requests.post('https://httpretty.example.com/api', data='{"nothing": "here"}
↪', headers={'Content-Type': 'application/json'})

assert httpretty.latest_requests[-1].url == 'https://httpbin.org/ip'
```

Parameters

- **method** – one of `httpretty.GET`, `httpretty.PUT`, `httpretty.POST`, `httpretty.DELETE`, `httpretty.HEAD`, `httpretty.PATCH`, `httpretty.OPTIONS`, `httpretty.CONNECT`

- **uri** – a string or regex pattern (e.g.: **"https://httpbin.org/ip"**)
- **body** – a string, defaults to `{"message": "HTTPretty :)"}`
- **adding_headers** – dict - headers to be added to the response
- **forcing_headers** – dict - headers to be forcefully set in the response
- **status** – an integer, defaults to **200**
- **responses** – a list of entries, ideally each created with *Response()*
- **priority** – an integer, useful for setting higher priority over previously registered urls. defaults to zero
- **match_querystring** – bool - whether to take the querystring into account when matching an URL
- **headers** – headers to be added to the response

> **Warning:** When using a port in the request, add a trailing slash if no path is provided otherwise Httpretty will not catch the request. Ex: `httpretty.register_uri(httpretty.GET, 'http://fakeuri.com:8080/', body='{"hello":"world"}')`

classmethod reset()
resets the internal state of HTTPretty, unregistering all URLs

httpretty.core.**set_default_thread_timeout**(*timeout*)
sets the default thread timeout for HTTPretty threads

Parameters **timeout** – int

httpretty.core.**url_fix**(*s*, *charset=None*)
escapes special characters

7.2 Http

httpretty.http.**last_requestline**(*sent_data*)
Find the last line in sent_data that can be parsed with parse_requestline

httpretty.http.**parse_requestline**(*s*)
http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5

```
>>> parse_requestline('GET / HTTP/1.0')
('GET', '/', '1.0')
>>> parse_requestline('post /testurl htTP/1.1')
('POST', '/testurl', '1.1')
>>> parse_requestline('Im not a RequestLine')
Traceback (most recent call last):
    ...
ValueError: Not a Request-Line
```

7.3 Utils

7.4 Exceptions

exception httpretty.errors.**HTTPrettyError**

exception httpretty.errors.**UnmockedError**(*message='Failed to handle network request'*, *request=None*, *address=None*)

CHAPTER

EIGHT

HACKING ON HTTPRETTY

8.1 install development dependencies

Note: HTTPretty uses GNU Make as default build tool.

```
make dependencies
```

8.2 next steps

1. run the tests with make:

```
make tests
```

2. hack at will
3. commit, push etc
4. send a pull request

CHAPTER

NINE

LICENSE

```
<HTTPretty - HTTP client mock for Python>
Copyright (C) <2011-2021> Gabriel Falcão <gabriel@nacaolivre.org>

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

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

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

CHAPTER

TEN

MAIN CONTRIBUTORS

HTTPretty has received many contributions but some folks made remarkable contributions and deserve extra credit:

- Andrew Gross ~> @andrewgross
- Hugh Saunders ~> @hughsaunders
- James Rowe ~> @JNRowe
- Matt Luongo ~> @mhluongo
- Steve Pulec ~> @spulec

CHAPTER

ELEVEN

RELEASE NOTES

11.1 Release 1.1.3

- Bugfix: #430 Respect socket timeout.

11.2 Release 1.1.2

- Bugfix: #426 Segmentation fault when running against a large amount of tests with `pytest --mypy`.

11.3 Release 1.1.1

- Bugfix: *httpretty.disable()* injects pyopenssl into `urllib3` even if it originally wasn't #417
- Bugfix: "Incompatibility with boto3 S3 put_object" #416
- Bugfix: "Regular expression for URL -> TypeError: wrap_socket() missing 1 required" #413
- Bugfix: "Making requests to non-stadard port throws TimeoutError "#387

11.4 Release 1.1.0

- Feature: Display mismatched URL within `UnmockedError` whenever possible. #388
- Feature: Display mismatched URL via logging. #419
- Add new properties to *`httpretty.core.HTTPrettyRequest`* (`protocol`, `host`, `url`, `path`, `method`).

Example usage:

```
import httpretty
import requests

@httpretty.activate(verbose=True, allow_net_connect=False)
def test_mismatches():
    requests.get('http://sql-server.local')
    requests.get('https://redis.local')
```

11.5 Release 1.0.5

- Bugfix: Support socket.socketpair() . #402
- Bugfix: Prevent exceptions from re-applying monkey patches. #406

11.6 Release 1.0.4

- Python 3.8 and 3.9 support. #407

11.7 Release 1.0.3

- Fix compatibility with urllib3>=1.26. #410

11.8 Release 1.0.0

- Drop Python 2 support.
- Fix usage with redis and improve overall real-socket passthrough. #271.
- Fix TypeError: wrap_socket() missing 1 required positional argument: 'sock' (#393)
- Merge pull request #364
- Merge pull request #371
- Merge pull request #379
- Merge pull request #386
- Merge pull request #302
- Merge pull request #373
- Merge pull request #383
- Merge pull request #385
- Merge pull request #389
- Merge pull request #391
- Fix simple typo: neighter -> neither.
- Updated documentation for register_uri concerning using ports.
- Clarify relation between `enabled` and `httprettized` in API docs.
- Align signature with builtin socket.

11.9 Release 0.9.4

Improvements:

- Official Python 3.6 support
- Normalized coding style to comform with PEP8 (partially)
- Add more API reference coverage in docstrings of members such as *httpretty.core.Entry*
- Continuous Integration building python 2.7 and 3.6
- Migrate from pip to pipenv

11.10 Release 0.8.4

Improvements:

- Refactored `core.py` and increased its unit test coverage to 80%. HTTPretty is slightly more robust now.

Bug fixes:

- POST requests being called twice #100

11.11 Release 0.6.5

Applied pull requests:

- continue on EAGAIN socket errors: #102 by kouk.
- Fix `fake_gethostbyname` for requests 2.0: #101 by mgood
- Add a way to match the querystrings: #98 by ametaireau
- Use common string case for URIInfo hostname comparison: #95 by mikewaters
- Expose httpretty.reset() to public API: #91 by imankulov
- Don't duplicate http ports number: #89 by mardiros
- Adding parsed_body parameter to simplify checks: #88 by toumorokoshi
- Use the real socket if it's not HTTP: #87 by mardiros

11.12 Release 0.6.2

- Fixing bug of lack of trailing slashes #73
- Applied pull requests #71 and #72 by @andresriancho
- Keyword arg coercion fix by @dupuy
- @papaeye fixed content-length calculation.

11.13 Release 0.6.1

- New API, no more camel case and everything is available through a simple import:

```
import httpretty

@httpretty.activate
def test_function():
    # httpretty.register_uri(...)
    # make request...
    pass
```

- Re-organized module into submodules

11.14 Release 0.5.14

- Delegate calls to other methods on socket
- Normalized header strings
- Callbacks are more intelligent now
- Normalize urls matching for url quoting

11.15 Release 0.5.12

- HTTPretty doesn't hang when using other application protocols under a @httprettified decorated test.

11.16 Release 0.5.11

- Ability to know whether HTTPretty is or not enabled through `httpretty.is_enabled()`

11.17 Release 0.5.10

- Support to multiple methods per registered URL. Thanks @hughsaunders

11.18 Release 0.5.9

- Fixed python 3 support. Thanks @spulec

11.19 Release 0.5.8

- Support to *register regular expressions to match urls*
- *Body callback* suppport
- Python 3 support

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

N

P

Q

R

S

U

V