pook Documentation

Release 0.2.5

Tomas Aparicio

Oct 30, 2018

Contents

CHAPTER 1

Installation

1.1 PyPI

You can install the last stable release of Expects from PyPI using pip or easy_install:

```
$ pip install pook
```

1.2 GitHub

Or install the latest sources from Github:

```
$ pip install -e git+git://github.com/h2non/pook.git#egg=pook
```

Also you can download a source code package from Github and install it using setuptools:

```
$ tar xvf pook-{version}.tar.gz
$ cd pook
$ python setup.py install
```

CHAPTER 2

How it works

2.1 HTTP traffic interception

In a nutshell, `pook` uses `unittest.mock` standard Python package in order to patch external library objects, allowing `pook` HTTP interceptor adapter to patch any third-party packages and intercept specific function calls.

`pook` entirely relies on this interception strategy, therefore in the meantime `pook` is active, any outgoing HTTP traffic intercepted by the supported HTTP clients won't imply any real TCP networking, except if you enabled the real networking mode via `pook.enable_network()`, which in that case real network traffic can be established.

Worth clarifying that `pook` only works at Python programmatic runtime level. There's no network/socket programming involved.

2.2 HTTP request matching

Outgoing HTTP traffic is intercepted and matched against a pool of mock matchers defined in your mock expectations.

Matching process in sequential and executed as FIFO order, meaning the first has always preference.

For instance, if you declare multiple identical mocks, the first one will be matched first and the others will be ignored. Once the first one expires, the subsequent mock definition in the chain will be matched instead.

2.3 Real networking mode

By default real networking mode is disabled. This basically means that real networking will not happen unless you explicitely enable it.

This behaviour has been adopted to improve predictability, control and mitigate developers fear between behaviour boundaries and expectations.

`pook` will prevent you to communicate with real HTTP servers unless you enable it via: `pook.enable_network()`.

Also, you can partially restrict the real networking by filtering only certain hosts. You can do that via `pook.use_network_filter(filter_fn)`.

2.4 Volatile vs Persistent mocks

By default, mocks are volatile. This means that once a mock has been matched, and therefore consumed, it will be flushed.

You can modify this behaviour via:

Explicitly defininging the TTL of each mock, so effectively the number of times the mock can be matched and consumed:

```
# Match a mock up to 5 times, then flush it
pook.get('server.com/api').times(5)

# The above is equivalent to
pook.get('server.com/api', times=5)
```

Explicitly defininging a persistent mock:

```
# Make a mock definition persistent, so it won't be never flushed
pook.get('server.com/api').persist()

# The above is equivalent to
pook.get('server.com/api', persist=True)
```

CHAPTER 3

Examples

3.1 Basic mocking example using requests

```
import pook
import requests


# Enable mock engine
pook.on()

pook.get('httpbin.org/ip',
         reply=403, response_type='json',
         response_headers={'server': 'pook'},
         response_json={'error': 'not found'})

pook.get('httpbin.org/headers',
         reply=404, response_type='json',
         response_headers={'server': 'pook'},
         response_json={'error': 'not found'})

res = requests.get('http://httpbin.org/ip')
print('Status:', res.status_code)
print('Headers:', res.headers)
print('Body:', res.json())

res = requests.get('http://httpbin.org/headers')
print('Status:', res.status_code)
print('Headers:', res.headers)
print('Body:', res.json())

print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())
print('Unmatched requests:', pook.unmatched_requests())
```

3.2 Chainable API DSL

```
import json
import pook
import requests


# Enable mock engine
pook.on()

(pook.post('httpbin.org/post')
    .json({'foo': 'bar'})
    .type('json')
    .header('Client', 'requests')
    .reply(204)
    .headers({'server': 'pook'})
    .json({'error': 'simulated'}))

res = requests.post('http://httpbin.org/post',
                    data=json.dumps({'foo': 'bar'}),
                    headers={'Client': 'requests',
                             'Content-Type': 'application/json'})

print('Status:', res.status_code)
print('Body:', res.json())

print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())
```

3.3 Context manager for isolated mocking

```
import pook
import requests


with pook.use():
    pook.get('httpbin.org/ip', reply=403,
             response_headers={'pepe': 'lopez'},
             response_json={'error': 'not found'})

    res = requests.get('http://httpbin.org/ip')
    print('Status:', res.status_code)
    print('Headers:', res.headers)
    print('Body:', res.json())

    print('Is done:', pook.isdone())
    print('Pending mocks:', pook.pending_mocks())
    print('Unmatched requests:', pook.unmatched_requests())
```

3.4 Single mock context manager definition for isolated mocking

```
import pook
import requests


# Define a new mock that will be only active within the context manager
with pook.get('httpbin.org/ip', reply=403,
              response_headers={'pepe': 'lopez'},
              response_json={'error': 'not found'}) as mock:

    res = requests.get('http://httpbin.org/ip')
    print('#1 Status:', res.status_code)
    print('#1 Headers:', res.headers)
    print('#1 Body:', res.json())
    print('-----------------')

    res = requests.get('http://httpbin.org/ip')
    print('#2 Status:', res.status_code)
    print('#2 Headers:', res.headers)
    print('#2 Body:', res.json())
    print('-----------------')

    print('Mock is done:', mock.isdone())
    print('Mock matches:', mock.total_matches)

    print('Is done:', pook.isdone())
    print('Pending mocks:', pook.pending_mocks())
    print('Unmatched requests:', pook.unmatched_requests())


# Explicitly disable mock engine
pook.off()

# Perform a real HTTP request since we are running the
# request outside of the context manager
res = requests.get('http://httpbin.org/ip')
print('#3 Status:', res.status_code)
print('#3 Headers:', res.headers)
print('#3 Body:', res.json())
```

3.5 Declaring mocks as decorators

```
import pook
import requests


@pook.get('http://httpbin.org/status/500', reply=204)
@pook.get('http://httpbin.org/status/400', reply=200, persist=True)
def fetch(url):
    return requests.get(url)


# Test function
```

(continues on next page)

(continued from previous page)

```
res = fetch('http://httpbin.org/status/400')
print('#1 status:', res.status_code)

res = fetch('http://httpbin.org/status/500')
print('#2 status:', res.status_code)

print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())

# Disable mock engine
pook.off()

# Test real request
res = requests.get('http://httpbin.org/status/400')
print('Test status:', res.status_code)
```

3.6 Featured JSON body matching

```
import json
import pook
import requests

# Enable mock engine
pook.on()

(pook.post('httpbin.org/post')
    .json({'foo': 'bar'})
    .reply(204)
    .json({'error': 'simulated'}))

res = requests.post('http://httpbin.org/post',
                    data=json.dumps({'foo': 'bar'}))
print('Status:', res.status_code)
print('Body:', res.json())

print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())
```

3.7 JSONSchema based body matching

```
import json
import pook
import requests


schema = {
    'type': 'object',
    'properties': {
        'foo': {'type': 'string'},
    }
}
```

(continues on next page)

(continued from previous page)

```
# Enable mock engine
pook.on()

(pook.post('httpbin.org/post')
    .jsonschema(schema)
    .reply(204)
    .json({'error': 'simulated'}))

res = requests.post('http://httpbin.org/post',
                    data=json.dumps({'foo': 'bar'}))

print('Status:', res.status_code)
print('Body:', res.json())

print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())
```

3.8 Enable real networking mode

```
import pook
import requests


# Enable mock engine
pook.on()

# Enable network mode
pook.enable_network()

(pook.get('httpbin.org/headers')
    .reply(204)
    .headers({'server': 'pook'})
    .json({'error': 'simulated'}))

res = requests.get('http://httpbin.org/headers')
print('Mock status:', res.status_code)

# Real network request, since pook cannot match any mock
res = requests.get('http://httpbin.org/ip')
print('Real status:', res.status_code)

print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())

# Disable network mode once we're done
pook.disable_network()
```

3.9 Persistent mock

```
import pook
import requests


# Enable mock engine
pook.on()

(pook.get('httpbin.org')
    .headers({'Client': 'requests'})
    .persist()
    .reply(400)
    .headers({'server': 'pook'})
    .json({'error': 'simulated'}))

res = requests.get('http://httpbin.org',
                   headers={'Client': 'requests'})

print('Status:', res.status_code)
print('Headers:', res.headers)
print('Body:', res.json())

print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())
```

3.10 Time TTL limited mock

```
import pook
import requests


# Enable mock engine
pook.on()

# Declare mock
(pook.get('httpbin.org')
    .times(2)
    .reply(400)
    .headers({'server': 'pook'})
    .json({'error': 'simulated'}))

# Mock request 1
res = requests.get('http://httpbin.org')
print('#1 status:', res.status_code)
print('#1 body:', res.json())

# Mock request 2
res = requests.get('http://httpbin.org')
print('#2 status:', res.status_code)
print('#2 body:', res.json())

# Real request 3
try:
```

(continues on next page)

(continued from previous page)

```
    requests.get('http://httpbin.org')
except:
    print('Request #3 not matched due to expired mock')

print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())
```

3.11 Regular expression matching

```
import pook
import requests


# Enable mock engine
pook.on()

# Mock definition based
(pook.get(pook.regex('h[t]{2}pbin.*'))
    .path(pook.regex('/foo/[a-z]+/baz'))
    .header('Client', pook.regex('requests|pook'))
    .times(2)
    .reply(200)
    .headers({'server': 'pook'})
    .json({'foo': 'bar'}))

# Perform request
res = requests.get('http://httpbin.org/foo/bar/baz',
                   headers={'Client': 'requests'})
print('Status:', res.status_code)
print('Body:', res.json())

# Perform second request
res = requests.get('http://httpbin.org/foo/foo/baz',
                   headers={'Client': 'pook'})
print('Status:', res.status_code)
print('Body:', res.json())

print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())
```

3.12 `unittest` integration

```
import pook
import unittest
import requests


class TestUnitTestEngine(unittest.TestCase):

    @pook.on
    def test_request(self):
```

(continues on next page)

(continued from previous page)

```
        pook.get('server.com/foo').reply(204)
        res = requests.get('http://server.com/foo')
        self.assertEqual(res.status_code, 204)

    def test_request_with_context_manager(self):
        with pook.use():
            pook.get('server.com/bar', reply=204)
            res = requests.get('http://server.com/bar')
            self.assertEqual(res.status_code, 204)

    @pook.on
    def test_no_match_exception(self):
        pook.get('server.com/bar', reply=204)
        try:
            requests.get('http://server.com/baz')
        except:
            pass
        else:
            raise RuntimeError('expected to fail')
```

3.13 `py.test` integration

```
# -*- coding: utf-8 -*-

import pook
import pytest
import requests


@pook.activate
def test_simple_pook_request():
    pook.get('server.com/foo').reply(204)
    res = requests.get('http://server.com/foo')
    assert res.status_code == 204


@pook.on
def test_enable_engine():
    pook.get('server.com/foo').reply(204)
    res = requests.get('http://server.com/foo')
    assert res.status_code == 204
    pook.disable()


@pook.get('server.com/bar', reply=204)
def test_decorator():
    res = requests.get('http://server.com/bar')
    assert res.status_code == 204


def test_context_manager():
    with pook.use():
        pook.get('server.com/baz', reply=204)
        res = requests.get('http://server.com/baz')
```

(continues on next page)

(continued from previous page)

```
        assert res.status_code == 204


@pook.on
def test_no_match_exception():
    pook.get('server.com/bar', reply=204)
    with pytest.raises(Exception):
        requests.get('http://server.com/baz')
```

3.14 Simulated error exception on mock matching

```
import pook
import requests


# Enable mock engine
pook.on()

# Simulated error exception on request matching
pook.get('httpbin.org/status/500', error=Exception('simulated error'))

try:
    requests.get('http://httpbin.org/status/500')
except Exception as err:
    print('Error:', err)
```

3.15 Using `urllib3` as HTTP client

```
import pook
import urllib3


# Mock HTTP traffic only in the given context
with pook.use():
    pook.get('http://httpbin.org/status/404').reply(204)

    # Intercept request
    http = urllib3.PoolManager()
    r = http.request('GET', 'http://httpbin.org/status/404')
    print('#1 status:', r.status)


# Real request outside of the context manager
http = urllib3.PoolManager()
r = http.request('GET', 'http://httpbin.org/status/404')
print('#2 status:', r.status)
```

3.16 Asynchronous HTTP request using `aiohttp`

```
# flake8: noqa

import pook
import aiohttp
import asyncio
import async_timeout


async def fetch(session, url, data):
    with async_timeout.timeout(10):
        async with session.get(url, data=data) as res:
            print('Status:', res.status)
            print('Headers:', res.headers)
            print('Body:', await res.text())


with pook.use(network=True):
    pook.get('http://httpbin.org/ip',
             reply=404, response_type='json',
             response_headers={'Server': 'nginx'},
             response_json={'error': 'not found'})

    async def main(loop):
        async with aiohttp.ClientSession(loop=loop) as session:
            await fetch(session, 'http://httpbin.org/ip',
                        bytearray('foo bar', 'utf-8'))

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main(loop))
```

3.17 Using `http.client` standard Python package as HTTP client

```
import http.client
import pook


# Enable mock engine
pook.on()

mock = pook.get('http://httpbin.org/ip',
                reply=404, response_type='json',
                response_json={'error': 'not found'})

conn = http.client.HTTPConnection('httpbin.org')
conn.request('GET', '/ip')

res = conn.getresponse()
print('Status:', res.status, res.reason)
print('Headers:', res.headers)
print('Body:', res.read())
print('Mock calls:', mock.calls)
```

(continues on next page)

(continued from previous page)

```
print('Is done:', pook.isdone())
print('Pending mocks:', pook.pending_mocks())
print('Unmatched requests:', pook.unmatched_requests())
```

3.18 Example using 'mocket'_ Python library as underlying mock engine

```
# Be sure you have `mocket` installed:
# $ pip install mocket

import pook
import requests
from mocket.plugins.pook_mock_engine import MocketEngine

# Use mocket library as underlying mock engine
pook.set_mock_engine(MocketEngine)

# Explicitly enable pook HTTP mocking (required)
pook.on()

# Target server URL to mock out
url = 'http://twitter.com/api/1/foobar'

# Define your mock
mock = pook.get(url,
                reply=404, times=2,
                headers={'content-type': 'application/json'},
                response_json={'error': 'foo'})

# Run first HTTP request
requests.get(url)
assert mock.calls == 1

# Run second HTTP request
res = requests.get(url)
assert mock.calls == 2

# Assert response data
assert res.status_code == 404
assert res.json() == {'error': 'foo'}

# Explicitly disable pook (optional)
pook.off()
```

3.19 Hy programming language example

```
(import [pook])
(import [requests])

(defn request [url &optional [status 404]]
```

(continues on next page)

(continued from previous page)

```
  (doto (.mock pook url) (.reply status))
  (let [res (.get requests url)]
    (. res status_code)))

(defn run []
  (with [(.use pook)]
    (print "Status:" (request "http://foo.com/bar" :status 204))))

;; Run test program
(defmain [&args] (run))
```

CHAPTER 4

API Documentation

4.1 Core API

pook.**activate**(*fn=None*)

Enables the HTTP traffic interceptors.

This function can be used as decorator.

Parameters **fn** (*function*) – Optional function argument if used as decorator.

Returns decorator wrapper function, only if called as decorator.

Return type function

Example:

```
# Standard use case
pook.activate()
pook.mock('server.com/foo').reply(404)

res = requests.get('server.com/foo')
assert res.status_code == 404
pook.disable()

# Decorator use case
@pook.activate
def test_request():
    pook.mock('server.com/foo').reply(404)

    res = requests.get('server.com/foo')
    assert res.status_code == 404
```

pook.**on**(*fn=None*)

Enables the HTTP traffic interceptors. Alias to `pook.activate()`.

Parameters **fn** (*function*) – Optional function argument if used as decorator.

Returns decorator wrapper function, only if called as decorator.

Return type function

Example:

```
# Standard usage
pook.on()
pook.mock('server.com/foo').reply(404)

res = requests.get('server.com/foo')
assert res.status_code == 404

# Usage as decorator
@pook.on
def test_request():
    pook.mock('server.com/foo').reply(404)

    res = requests.get('server.com/foo')
    assert res.status_code == 404
```

pook.**disable**()
Disables HTTP traffic interceptors.

pook.**off**()
Disables mock engine, HTTP traffic interceptors and flushed all the registered mocks.

Internally, it calls `pook.disable()` and `pook.off()`.

pook.**reset**()
Resets current mock engine state, flushing all the registered mocks.

This action will not disable the mock engine.

pook.**engine**()
Returns the current running mock engine.

Returns current used engine.

Return type *pook.Engine*

pook.**use_network**()
Creates a new mock engine to be used as context manager

Example:

```
with pook.use_network() as engine:
    pook.mock('server.com/foo').reply(404)

    res = requests.get('server.com/foo')
    assert res.status_code == 404
```

pook.**enable_network**(**hostnames*)
Enables real networking mode for unmatched mocks in the current mock engine.

pook.**disable_network**()
Disables real traffic networking mode in the current mock engine.

pook.**get**(*url*, ***kw*)
Registers a new mock HTTP request with GET method.

Parameters

- **url** (*str*) – request URL to mock.
- **activate** (*bool*) – force mock engine activation. Defaults to `False`.
- ****kw** (*mixed*) – variadic arguments to `pook.Mock` constructor.

Returns mock instance

Return type *pook.Mock*

`pook.post`(*url*, ***kw*)
Registers a new mock HTTP request with POST method.

Parameters

- **url** (*str*) – request URL to mock.
- **activate** (*bool*) – force mock engine activation. Defaults to `False`.
- ****kw** (*mixed*) – variadic arguments to `pook.Mock` constructor.

Returns mock instance

Return type *pook.Mock*

`pook.put`(*url*, ***kw*)
Registers a new mock HTTP request with PUT method.

Parameters

- **url** (*str*) – request URL to mock.
- **activate** (*bool*) – force mock engine activation. Defaults to `False`.
- ****kw** (*mixed*) – variadic arguments to `pook.Mock` constructor.

Returns mock instance

Return type *pook.Mock*

`pook.patch`(*url=None*, ***kw*)
Registers a new mock HTTP request with PATCH method.

Parameters

- **url** (*str*) – request URL to mock.
- **activate** (*bool*) – force mock engine activation. Defaults to `False`.
- ****kw** (*mixed*) – variadic arguments to `pook.Mock` constructor.

Returns new mock instance.

Return type *pook.Mock*

`pook.head`(*url*, ***kw*)
Registers a new mock HTTP request with HEAD method.

Parameters

- **url** (*str*) – request URL to mock.
- **activate** (*bool*) – force mock engine activation. Defaults to `False`.
- ****kw** (*mixed*) – variadic arguments to `pook.Mock` constructor.

Returns mock instance

Return type *pook.Mock*

pook.**use**(*network=False*)
Creates a new isolated mock engine to be used via context manager.

Example:

```
with pook.use() as engine:
    pook.mock('server.com/foo').reply(404)

    res = requests.get('server.com/foo')
    assert res.status_code == 404
```

pook.**set_mock_engine**(*engine*)
Sets a custom mock engine, replacing the built-in one.

This is particularly useful if you want to replace the built-in HTTP traffic mock interceptor engine with your custom one.

For mock engine implementation details, see *pook.MockEngine*.

Parameters **engine** (pook.MockEngine) – custom mock engine to use.

pook.**delete**(*url*, ***kw*)
Registers a new mock HTTP request with DELETE method.

Parameters

- **url** (*str*) – request URL to mock.
- **activate** (*bool*) – force mock engine activation. Defaults to False.
- ****kw** (*mixed*) – variadic arguments to pook.Mock constructor.

Returns mock instance

Return type *pook.Mock*

pook.**options**(*url=None*, ***kw*)
Registers a new mock HTTP request with OPTIONS method.

Parameters

- **url** (*str*) – request URL to mock.
- **activate** (*bool*) – force mock engine activation. Defaults to False.
- ****kw** (*mixed*) – variadic arguments to pook.Mock constructor.

Returns new mock instance.

Return type *pook.Mock*

pook.**pending**()
Returns the numbers of pending mocks to be matched.

Returns number of pending mocks to match.

Return type int

pook.**ispending**()
Returns the numbers of pending mocks to be matched.

Returns number of pending mocks to match.

Return type int

pook.**mock**(*url=None*, ***kw*)
Creates and register a new HTTP mock.

Parameters

- **url** (*str*) – request URL to mock.
- **activate** (*bool*) – force mock engine activation. Defaults to `False`.
- ****kw** (*mixed*) – variadic keyword arguments.

Returns mock instance

Return type *pook.Mock*

`pook.pending_mocks()`
Returns pending mocks to be matched.

Returns pending mock instances.

Return type list

`pook.unmatched_requests()`
Returns a `tuple` of unmatched requests.

Unmatched requests will be registered only if `networking` mode has been enabled.

Returns unmatched intercepted requests.

Return type list

`pook.isunmatched()`
Returns `True` if there are unmatched requests. Otherwise `False`.

Unmatched requests will be registered only if `networking` mode has been enabled.

Returns bool

`pook.unmatched()`
Returns the total number of unmatched requests intercepted by pook.

Unmatched requests will be registered only if `networking` mode has been enabled.

Returns total number of unmatched requests.

Return type int

`pook.isactive()`
Returns `True` if pook is active and intercepting traffic. Otherwise `False`.

Returns True is all the registered mocks are gone, otherwise False.

Return type bool

`pook.isdone()`
Returns True if all the registered mocks has been triggered.

Returns True is all the registered mocks are gone, otherwise False.

Return type bool

`pook.regex(expression, flags=2)`
Convenient shortcut to `re.compile()` for fast, easy to use regular expression compilation without an extra import statement.

Parameters

- **expression** (*str*) – regular expression value.
- **flags** (*int*) – optional regular expression flags. Defaults to `re.IGNORECASE`

Returns string based regular expression.

Return type expression (str)

Raises `Exception` – in case of regular expression compilation error

Example:

```
(pook
    .get('api.com/foo')
    .header('Content-Type', pook.regex('[a-z]{1,4}')))
```

class pook.**Engine**(*network=False*)

Bases: object

Engine represents the mock interceptor and matcher engine responsible of triggering interceptors and match outgoing HTTP traffic.

Parameters **network** (*bool, optional*) – enables/disables real networking mode.

debug
bool – enables/disables debug mode.

active
bool – stores the current engine activation status.

networking
bool – stores the current engine networking mode status.

mocks
list[pook.Mock] – stores engine mocks.

filters
list[function] – stores engine-level mock filter functions.

mappers
list[function] – stores engine-level mock mapper functions.

interceptors
list[pook.BaseInterceptor] – stores engine-level HTTP traffic interceptors.

unmatched_reqs
list[pook.Request] – stores engine-level unmatched outgoing HTTP requests.

network_filters
list[function] – stores engine-level real networking mode filters.

activate()
Activates the registered interceptors in the mocking engine.

This means any HTTP traffic captures by those interceptors will trigger the HTTP mock matching engine in order to determine if a given HTTP transaction should be mocked out or not.

add_interceptor(**interceptors*)
Adds one or multiple HTTP traffic interceptors to the current mocking engine.

Interceptors are typically HTTP client specific wrapper classes that implements the pook interceptor interface.

Note: this method is may not be implemented if using a custom mock engine.

Parameters **interceptors** (pook.interceptors.BaseInterceptor) –

add_mock(*mock*)
Adds a new mock instance to the current engine.

Parameters **mock** (pook.Mock) – mock instance to add.

disable()
Disables interceptors and stops intercepting any outgoing HTTP traffic.

disable_network()
Disables real networking mode.

enable_network(**hostnames*)
Enables real networking mode, optionally passing one or multiple hostnames that would be used as filter.

If at least one hostname matches with the outgoing traffic, the request will be executed via the real network.

Parameters ***hostnames** – optional list of host names to enable real network against them. hostname value can be a regular expression.

filter(**filters*)
Append engine-level HTTP request filter functions.

Parameters **filters*** – variadic filter functions to be added.

flush_interceptors()
Flushes registered interceptors in the current mocking engine.

This method is low-level. Only call it if you know what you are doing.

Note: this method is may not be implemented if using a custom mock engine.

flush_mocks()
Flushes the current mocks.

flush_network_filters()
Flushes registered real networking filters in the current mock engine.

isactive()
Returns the current engine enabled/disabled status.

Returns `True` if the engine is active. Otherwise `False`.

Return type bool

isdone()
Returns True if all the registered mocks has been triggered.

Returns True is all the registered mocks are gone, otherwise False.

Return type bool

ispending()
Returns the `True` if the engine has pending mocks to be matched. Otherwise `False`.

Returns bool

isunmatched()
Returns `True` if there are unmatched requests. Otherwise `False`.

Unmatched requests will be registered only if `networking` mode has been enabled.

Returns bool

map(**mappers*)
Append engine-level HTTP request mapper functions.

Parameters **filters*** – variadic mapper functions to be added.

match(*request*)
Matches a given Request instance contract against the registered mocks.

If a mock passes all the matchers, its response will be returned.

Parameters **request** (pook.Request) – Request contract to match.

Raises pook.PookNoMatches – if networking is disabled and no mock matches with the given request contract.

Returns the mock response to be used by the interceptor.

Return type *pook.Response*

mock(*url=None*, ***kw*)

Creates and registers a new HTTP mock in the current engine.

Parameters

- **url** (*str*) – request URL to mock.
- **activate** (*bool*) – force mock engine activation. Defaults to False.
- ****kw** (*mixed*) – variadic keyword arguments for Mock constructor.

Returns new mock instance.

Return type *pook.Mock*

pending()

Returns the number of pending mocks to be matched.

Returns number of pending mocks.

Return type int

pending_mocks()

Returns a tuple of pending mocks to be matched.

Returns pending mock instances.

Return type tuple

remove_interceptor(*name*)

Removes a specific interceptor by name.

Note: this method is may not be implemented if using a custom mock engine.

Parameters **name** (*str*) – interceptor name to disable.

Returns *True* if the interceptor was disabled, otherwise *False*.

Return type bool

remove_mock(*mock*)

Removes a specific mock instance by object reference.

Parameters **mock** (pook.Mock) – mock instance to remove.

reset()

Resets and flushes engine state and mocks to defaults.

set_mock_engine(*engine*)

Sets a custom mock engine, replacing the built-in one.

This is particularly useful if you want to replace the built-in HTTP traffic mock interceptor engine with your custom one.

For mock engine implementation details, see *pook.MockEngine*.

Parameters **engine** (pook.MockEngine) – custom mock engine to use.

should_use_network(*request*)
Verifies if real networking mode should be used for the given request, passing it to the registered network filters.

Parameters **request** (pook.Request) – outgoing HTTP request to test.

Returns bool

unmatched()
Returns the total number of unmatched requests intercepted by pook.

Unmatched requests will be registered only if `networking` mode has been enabled.

Returns total number of unmatched requests.

Return type int

unmatched_requests()
Returns a `tuple` of unmatched requests.

Unmatched requests will be registered only if `networking` mode has been enabled.

Returns unmatched intercepted requests.

Return type list

use_network_filter(**fn*)
Adds network filters to determine if certain outgoing unmatched HTTP traffic can stablish real network connections.

Parameters ***fn** (*function*) – variadic function filter arguments to be used.

class pook.**Mock**(*request=None*, *response=None*, ***kw*)
Bases: object

Mock is used to declare and compose the HTTP request/response mock definition and matching expectations, which provides fluent API DSL.

Parameters

- **url** (*str*) – URL to match. E.g: `server.com/api?foo=bar`.
- **method** (*str*) – HTTP method name to match. E.g: `GET`.
- **path** (*str*) – URL path to match. E.g: `/api/users`.
- **headers** (*dict*) – Header values to match. E.g: `{'server': 'nginx'}`.
- **header_present** (*str*) – Matches is a header is present.
- **headers_present** (*list|tuple*) – Matches if multiple headers are present.
- **type** (*str*) – Matches MIME `Content-Type` header. E.g: `json`, `xml`, `html`, `text/plain`
- **content** (*str*) – Same as `type` argument.
- **params** (*dict*) – Matches the given URL params.
- **param_exists** (*str*) – Matches if a given URL param exists.
- **params_exists** (*list|tuple*) – Matches if a given URL params exists.
- **body** (*str|regex*) – Matches the payload body by regex or strict comparison.
- **json** (*dict|list|str|regex*) – Matches the payload body against the given JSON or regular expression.

- **jsonschema** (*dict|str*) – Matches the payload body against the given JSONSchema.
- **xml** (*str|regex*) – matches the payload body against the given XML string or regular expression.
- **file** (*str*) – Disk file path to load body from. Analog to `body` param.
- **times** (*int*) – Mock TTL or maximum number of times that the mock can be matched.
- **persist** (*bool*) – Enable persistent mode. Mock won't be flushed even if it matched one or multiple times.
- **delay** (*int*) – Optional network delay simulation (only applicable when using `aiohttp` HTTP client).
- **callback** (*function*) – optional callback function called every time the mock is matched.
- **reply** (*int*) – Mock response status. Defaults to `200`.
- **response_status** (*int*) – Mock response status. Alias to `reply` param.
- **response_headers** (*dict*) – Response headers to use.
- **response_type** (*str*) – Response MIME type expression or alias. Analog to `type` param. E.g: `json`, `xml`, `text/plain`.
- **response_body** (*str*) – Response body to use.
- **response_json** (*dict|list|str*) – Response JSON to use. If Python is passed, it will be serialized as JSON transparently.
- **response_xml** (*str*) – XML body string to use.
- **request** (`pook.Request`) – Optional. Request mock definition object.
- **response** (`pook.Response`) – Optional. Response mock definition object.

Returns pook.Mock

add_matcher(*matcher*)

Adds one or multiple custom matchers instances.

Matchers must implement the following interface:

- `.__init__(expectation)`
- `.match(request)`
- `.name = str`

Matchers can optionally inherit from `pook.matchers.BaseMatcher`.

Parameters ***matchers** (*pook.matchers.BaseMatcher*) – matchers to add.

Returns current Mock instance.

Return type self

body(*body*)

Defines the body data to match.

`body` argument can be a `str`, `binary` or a regular expression.

Parameters **body** (*str|binary|regex*) – body data to match.

Returns current Mock instance.

Return type self

callback(**callbacks*)
Registers one or multiple callback that will be called every time the current mock matches an outgoing HTTP request.

Parameters ***callbacks** (*function*) – callback functions to call.

Returns current Mock instance.

Return type self

calls
Accessor to retrieve the amount of mock matched calls.

Returns int

content(*value*)
Defines the `Content-Type` outgoing header value to match.

You can pass one of the following type aliases instead of the full MIME type representation:

- `json` = `application/json`
- `xml` = `application/xml`
- `html` = `text/html`
- `text` = `text/plain`
- `urlencoded` = `application/x-www-form-urlencoded`
- `form` = `application/x-www-form-urlencoded`
- `form-data` = `application/x-www-form-urlencoded`

Parameters **value** (*str*) – type alias or header value to match.

Returns current Mock instance.

Return type self

delay(*delay=1000*)
Delay network response with certain milliseconds. Only supported by asynchronous HTTP clients, such as `aiohttp`.

Parameters **delay** (*int*) – milliseconds to delay response.

Returns current Mock instance.

Return type self

done
Attribute accessor that would be `True` if the current mock is done, and therefore have been matched multiple times.

Returns bool

error(*error*)
Defines a simulated exception error that will be raised.

Parameters **error** (*str|Exception*) – error to raise.

Returns current Mock instance.

Return type self

file(*path*)
Reads the body to match from a disk file.

Parameters **path** (*str*) – relative or absolute path to file to read from.

Returns current Mock instance.

Return type self

filter(**filters*)
Registers one o multiple request filters used during the matching phase.

Parameters ***mappers** (*function*) – variadic mapper functions.

Returns current Mock instance.

Return type self

header(*name*, *value*)
Defines a URL path to match.

Only call this method if the URL has no path already defined.

Parameters **path** (*str*) – URL path value to match. E.g: `/api/users`.

Returns current Mock instance.

Return type self

header_present(**names*)
Defines a new header matcher expectation that must be present in the outgoing request in order to be satisfied, no matter what value it hosts.

Header keys are case insensitive.

Parameters ***names** (*str*) – header or headers names to match.

Returns current Mock instance.

Return type self

Example:

```
(pook.get('server.com/api')
    .header_present('content-type'))
```

headers(*headers=None*, ***kw*)
Defines a dictionary of arguments.

Header keys are case insensitive.

Parameters

- **headers** (*dict*) – headers to match.
- ****headers** (*dict*) – headers to match as variadic keyword arguments.

Returns current Mock instance.

Return type self

headers_present(*headers*)
Defines a list of headers that must be present in the outgoing request in order to satisfy the matcher, no matter what value the headers hosts.

Header keys are case insensitive.

Parameters **headers** (*list|tuple*) – header keys to match.

Returns current Mock instance.

Return type self

Example:

```
(pook.get('server.com/api')
    .headers_present(['content-type', 'Authorization']))
```

isdone()
Returns `True` if the mock has been matched by outgoing HTTP traffic.

Returns `True` if the mock was matched succesfully.

Return type bool

ismatched()
Returns `True` if the mock has been matched at least once time.

Returns bool

json(*json*)
Defines the JSON body to match.

`json` argument can be an JSON string, a JSON serializable Python structure, such as a `dict` or `list` or it can be a regular expression used to match the body.

Parameters **json** (*str | dict | list | regex*) – body JSON to match.

Returns current Mock instance.

Return type self

jsonschema(*schema*)
Defines a JSONSchema representation to be used for body matching.

Parameters **schema** (*str | dict*) – dict or JSONSchema string to use.

Returns current Mock instance.

Return type self

map(**mappers*)
Registers one o multiple request mappers used during the mapping phase.

Parameters ***mappers** (*function*) – variadic mapper functions.

Returns current Mock instance.

Return type self

match(*request*)
Matches an outgoing HTTP request against the current mock matchers.

This method acts like a delegator to *pook.MatcherEngine*.

Parameters **request** (`pook.Request`) – request instance to match.

Raises `Exception` – if the mock has an exception defined.

Returns

True if the mock matches the outgoing HTTP request, otherwise `False`. Also returns an optional list of error exceptions.

Return type tuple(bool, list[Exception])

matched
Accessor property that would be `True` if the current mock have been matched at least once.

See `Mock.total_matches` for more information.

Returns bool

matches
Accessor to retrieve the mock match calls registry.

Returns list[MockCall]

method(*method*)
Defines the HTTP method to match. Use `*` to match any method.

Parameters **method** (*str*) – method value to match. E.g: `GET`.

Returns current Mock instance.

Return type self

param(*name*, *value*)
Defines an URL param key and value to match.

Parameters

- **name** (*str*) – param name value to match.
- **value** (*str*) – param name value to match.

Returns current Mock instance.

Return type self

param_exists(*name*)
Checks if a given URL param name is present in the URL.

Parameters **name** (*str*) – param name to check existence.

Returns current Mock instance.

Return type self

params(*params*)
Defines a set of URL query params to match.

Parameters **params** (*dict*) – set of params to match.

Returns current Mock instance.

Return type self

path(*path*)
Defines a URL path to match.

Only call this method if the URL has no path already defined.

Parameters **path** (*str*) – URL path value to match. E.g: `/api/users`.

Returns current Mock instance.

Return type self

persist(*status=None*)
Enables persistent mode for the current mock.

Returns current Mock instance.

Return type self

reply(*status=200*, ***kw*)
Defines the mock response.

Parameters

- **status** (*int, optional*) – response status code. Defaults to `200`.
- ****kw** (*dict*) – optional keyword arguments passed to `pook.Response` constructor.

Returns mock response definition instance.

Return type *pook.Response*

response(*status=200*, ***kw*)
Defines the mock response. Alias to `.reply()`

Parameters

- **status** (*int*) – response status code. Defaults to `200`.
- ****kw** (*dict*) – optional keyword arguments passed to `pook.Response` constructor.

Returns mock response definition instance.

Return type *pook.Response*

status(*code=200*)
Defines the response status code. Equivalent to `self.reply(code)`.

Parameters **code** (*int*) – response status code. Defaults to `200`.

Returns mock response definition instance.

Return type *pook.Response*

times(*times=1*)
Defines the TTL limit for the current mock.

The TTL number will determine the maximum number of times that the current mock can be matched and therefore consumed.

Parameters **times** (*int*) – TTL number. Defaults to `1`.

Returns current Mock instance.

Return type self

total_matches
Accessor property to retrieve the total number of times that the current mock has been matched.

Returns int

type(*value*)
Defines the request `Content-Type` header to match.

You can pass one of the following aliases instead of the full MIME type representation:

- `json` = `application/json`
- `xml` = `application/xml`
- `html` = `text/html`
- `text` = `text/plain`
- `urlencoded` = `application/x-www-form-urlencoded`
- `form` = `application/x-www-form-urlencoded`

- `form-data` = `application/x-www-form-urlencoded`

Parameters **value** (*str*) – type alias or header value to match.

Returns current Mock instance.

Return type self

url (*url*)
Defines the mock URL to match. It can be a full URL with path and query params.

Protocol schema is optional, defaults to `http://`.

Parameters **url** (*str*) – mock URL to match. E.g: `server.com/api`.

Returns current Mock instance.

Return type self

use (**matchers*)
Adds one or multiple custom matchers instances.

Matchers must implement the following interface:

- `.__init__(expectation)`
- `.match(request)`
- `.name = str`

Matchers can optionally inherit from `pook.matchers.BaseMatcher`.

Parameters ***matchers** (*pook.matchers.BaseMatcher*) – matchers to add.

Returns current Mock instance.

Return type self

xml (*xml*)
Defines a XML body value to match.

Parameters **xml** (*str | regex*) – body XML to match.

Returns current Mock instance.

Return type self

class pook.**Request**(*method='GET'*, ***kw*)
Bases: `object`

Request object representing the request mock expectation DSL.

Parameters

- **method** (*str*) – HTTP method to match. Defaults to `GET`.
- **url** (*str*) – URL request to intercept and match.
- **headers** (*dict*) – HTTP headers to match.
- **query** (*dict*) – URL query params to match. Complementely to URL defined query params.
- **body** (*str | regex*) – request body payload to match.
- **json** (*str | dict | list*) – JSON payload body structure to match.
- **xml** (*str*) – XML payload data structure to match.

method
str – HTTP method to match. Defaults to `GET`.

url
str – URL request to intercept and match.

headers
dict – HTTP headers to match.

query
dict – URL query params to match. Complementely to URL defined query params.

body
str|regex – request body payload to match.

json
str|dict|list – JSON payload body structure to match.

xml
str – XML payload data structure to match.

body

copy()
Copies the current Request object instance for side-effects purposes.

Returns copy of the current Request instance.

Return type *pook.Request*

extra

headers

json

keys = ('method', 'headers', 'body', 'url', 'query')

method

query

rawurl

url

xml

class pook.**Response**(***kw*)
Bases: `object`

Response is used to declare and compose an HTTP mock responses fields.

It provides a chainable DSL interface for easier and declarative usage.

Parameters

- **status** (*int*) – HTTP response status code. Defaults to `200`.
- **headers** (*dict*) – HTTP response headers.
- **body** (*str|bytes*) – HTTP response body.
- **json** (*str|dict|list*) – HTTP response JSON body.
- **xml** (*str*) – HTTP response XML body.
- **type** (*str*) – HTTP response content MIME type.

- **file** (*str*) – file path to HTTP body response.

mock
pook.Mock – reference to mock instance.

body(*body*)
Defines response body data.

Parameters **body** (*str|bytes*) – response body to use.

Returns `pook.Response` current instance.

Return type self

content(*name*)
Defines the response `Content-Type` header.

You can pass one of the following type aliases instead of the full MIME type representation:

- `json` = `application/json`
- `xml` = `application/xml`
- `html` = `text/html`
- `text` = `text/plain`
- `urlencoded` = `application/x-www-form-urlencoded`
- `form` = `application/x-www-form-urlencoded`
- `form-data` = `application/x-www-form-urlencoded`

Parameters **value** (*str*) – type alias or header value to match.

Returns `pook.Response` current instance.

Return type self

file(*path*)
Defines the response body from file contents.

Parameters **path** (*str*) – disk file path to load.

Returns `pook.Response` current instance.

Return type self

header(*key*, *value*)
Defines a new response header. Alias to `Response.header()`.

Parameters

- **header** (*str*) – header name.
- **value** (*str*) – header value.

Returns `pook.Response` current instance.

Return type self

headers(*headers*)
Defines a new response header. Alias to `Response.header()`.

Parameters

- **header** (*str*) – header name.

- **value** (*str*) – header value.

Returns pook.Response current instance.

Return type self

json(*data*)
Defines the mock response JSON body.

Parameters **data** (*dict | list | str*) – JSON body data.

Returns pook.Response current instance.

Return type self

mock
Getter accessor for *mock* attribute.

set(*header*, *value*)
Defines a new response header. Alias to Response.header().

Parameters

- **header** (*str*) – header name.
- **value** (*str*) – header value.

Returns pook.Response current instance.

Return type self

status(*code=200*)
Defines the response status code.

Parameters **code** (*int*) – response status code.

Returns pook.Response current instance.

Return type self

type(*name*)
Defines the response Content-Type header. Alias to Response.content(mime).

You can pass one of the following type aliases instead of the full MIME type representation:

- json = application/json
- xml = application/xml
- html = text/html
- text = text/plain
- urlencoded = application/x-www-form-urlencoded
- form = application/x-www-form-urlencoded
- form-data = application/x-www-form-urlencoded

Parameters **value** (*str*) – type alias or header value to match.

Returns pook.Response current instance.

Return type self

xml(*xml*)
Defines the mock response XML body.

For not it only supports `str` as input type.

Parameters **xml** (*str*) – XML body data to use.

Returns `pook.Response` current instance.

Return type self

class pook.**MatcherEngine**
Bases: `list`

HTTP request matcher engine used by *pook.Mock* to test if an intercepted outgoing HTTP request must be mocked out or not.

add(*matcher*)
Adds a new matcher function to the current engine.

Parameters **matcher** (*function*) – matcher function to be added.

flush()
Flushes the current matcher engine, removing all the registered matcher functions.

match(*request*)
Match the given HTTP request instance against the registered matcher functions in the current engine.

Parameters **request** (`pook.Request`) – outgoing request to match.

Returns

True if all matcher tests passes, otherwise `False`. Also returns an optional list of error exceptions.

Return type tuple(bool, list[Exception])

class pook.**MockEngine**(*engine*)
Bases: `object`

MockEngine implements the built-in *pook* mock engine based on HTTP interceptors strategy.

Mock engines must implementent the following methods:

- *engine.__init__(self, engine)*
- *engine.activate(self)*
- *engine.disable(self)*

Mock engines can optionally implement the follow methods:

- *engine.add_interceptors(self, *interceptors)*
- *engine.flush_interceptors(self)*
- *engine.disable_interceptor(self, name) -> bool*

Parameters **engine** (`pook.Engine`) – injected pook engine to be used.

engine
pook.Engine – stores pook engine to be used.

interceptors
list[pook.BaseInterceptor] – stores engine-level HTTP traffic interceptors.

activate()
Activates the registered interceptors in the mocking engine.

This means any HTTP traffic captures by those interceptors will trigger the HTTP mock matching engine in order to determine if a given HTTP transaction should be mocked out or not.

add_interceptor(**interceptors*)
Adds one or multiple HTTP traffic interceptors to the current mocking engine.

Interceptors are typically HTTP client specific wrapper classes that implements the pook interceptor interface.

Parameters **interceptors** (pook.interceptors.BaseInterceptor) –

disable()
Disables interceptors and stops intercepting any outgoing HTTP traffic.

flush_interceptors()
Flushes registered interceptors in the current mocking engine.

This method is low-level. Only call it if you know what you are doing.

remove_interceptor(*name*)
Removes a specific interceptor by name.

Parameters **name** (*str*) – interceptor name to disable.

Returns *True* if the interceptor was disabled, otherwise *False*.

Return type bool

4.2 Matchers API

4.3 Interceptors API

pook.interceptors.**add**(**interceptors*)
Registers a new HTTP client interceptor.

Parameters ***interceptors** (*interceptor*) – interceptor(s) to be added.

pook.interceptors.**get**(*name*)
Returns an interceptor by class name.

Parameters **name** (*str*) – interceptor class name or alias.

Returns found interceptor instance, otherwise `None`.

Return type interceptor

class pook.interceptors.**BaseInterceptor**(*engine*)
Bases: object

BaseInterceptor provides a base class for HTTP traffic interceptors implementations.

activate()
Activates the traffic interceptor. This method must be implemented by any interceptor.

disable()
Disables the traffic interceptor. This method must be implemented by any interceptor.

name
Exposes the interceptor class name.

class pook.interceptors.**Urllib3Interceptor**(*engine*)
Bases: pook.interceptors.base.BaseInterceptor

Urllib3 HTTP traffic interceptor.

activate()
Activates the traffic interceptor. This method must be implemented by any interceptor.

disable()
Disables the traffic interceptor. This method must be implemented by any interceptor.

class pook.interceptors.**HTTPClientInterceptor**(*engine*)
Bases: pook.interceptors.base.BaseInterceptor

urllib / http.client HTTP traffic interceptor.

activate()
Activates the traffic interceptor. This method must be implemented by any interceptor.

disable()
Disables the traffic interceptor. This method must be implemented by any interceptor.

class pook.interceptors.**AIOHTTPInterceptor**(*engine*)
Bases: pook.interceptors.base.BaseInterceptor

aiohttp HTTP client traffic interceptor.

activate()
Activates the traffic interceptor. This method must be implemented by any interceptor.

disable()
Disables the traffic interceptor. This method must be implemented by any interceptor.

CHAPTER 5

FAQ

5.1 How does it work?

Please, read how it works section.

5.2 What HTTP clients are supported?

Please, see supported HTTP clients section.

5.3 Does `pook` mock out all the outgoing HTTP traffic from my app?

Yes, that's the default behaviour: any outgoing HTTP traffic across the supported HTTP clients will be intercepted by `pook`.

In case that an outgoing traffic does not match any mock expectation, an exception error will be raised, telling you no mock was matched in order to review or fix your code accordingly.

You can for sure change this behaviour and don't raise any exception if no mock definition can be matched.

You can change this enabling the real networking mode via `pook.enable_network()`.

5.4 Can I use `pook` in a non testing environment?

Absolutely. `pook` is testing environment agnostic.

You simply have to take care of the side effects of mocking HTTP traffic in a runtime environment.

For that cases you probably want to enable the real networking mode.

5.5 Can I use pook with a custom HTTP traffic mock interceptor engine?

Yes, you can. `pook` is very modular and open for extensibility.

You can programmatically define the HTTP traffic mock engine you want to use via `pook.set_mock_engine(engine)`. This will replace the built-in one.

This can be particularly useful if you are already using another HTTP mocking engine that satisfy your needs, but you want to take benefit of `pook` features, versatility and simple to use expressive API.

For mock engine implementation details, see `pook.MockEngine` API documentation.

5.6 Can I use pook with any test framework?

Yes. `pook` is test framework agnostic. You can use it within `unittest`, `nosetests`, `pytest` or others.

CHAPTER 6

History

6.1 v0.2.5 / 2017-10-19

- refactor(setup): remove extra install dependency
- Fix py27 compatibility (#49)
- Add activate_async decorator (#48)
- fix typo in pook.mock.Mock.ismatched.__doc__ (#47)
- fix README example (#46)

6.2 v0.2.4 / 2017-10-03

- fix(#45): regex URL issue
- fix(travis): allow failures in pypy
- feat(docs): add sponsor banner
- refactor(History): normalize style

6.3 v0.2.3 / 2017-04-28

- feat(docs): add supported version for aiohttp
- Merge branch 'master' of https://github.com/h2non/pook
- fix(api): export missing symbol "disable_network"
- Update README.rst (#43)

6.4 v0.2.2 / 2017-04-03

- refactor(compare): disable maxDiff length limit while comparing values

6.5 v0.2.1 / 2017-03-25

- fix(engine): enable new mock engine on register if needed
- fix(engine): remove activate argument before instantiating the Mock

6.6 v0.2.0 / 2017-03-18

- refactor(engine): do not activate engine on mock declaration if not explicitly requested. This introduces a behavioral library change: you must explicitly use `pook.on()` to enable *pook* mock engine.

6.7 v0.1.14 / 2017-03-17

- feat(docs): list supported HTTP client versions
- fix(#41): disable mocks after decorator call invokation
- feat(examples): add mock context manager example file
- feat(#40): support context manager definitions
- feat(#39): improve unmatched request output
- feat(docs): add mocket example file
- feat(#33): add mocket examples and documentation

6.8 v0.1.13 / 2017-01-29

- fix(api): *mock.calls* property should be an *int*.

6.9 v0.1.12 / 2017-01-28

- feat(#33): proxy mock definitions into mock.Request
- refactor(api): *pook.unmatched_requests()* now returns a *list* instead of a lazy *tuple*.

6.10 v0.1.11 / 2017-01-14

- refactor(query)
- fix(#37): fix URL comparison
- fix(#38): proper mock engine interface validation.

6.11 v0.1.10 / 2017-01-13

- fix(#37): decode byte bodies
- feat(setup.py): add author email

6.12 v0.1.9 / 2017-01-06

- fix(Makefile): remove proper egg file
- feat(package): add wheel package distribution support
- feat(docs): add documentation links

6.13 v0.1.8 / 2016-12-24

- fix(assertion): extract regex pattern only when required
- feat(examples): add regular expression example

6.14 v0.1.7 / 2016-12-18

- feat(#33): add support for user defined custom mock engine

6.15 v0.1.6 / 2016-12-14

- fix(setup.py): force utf-8 encoding
- feat(setup.py): add encoding header
- feat(api): add debug mode
- refactor(docs): minor enhancements
- refactor(tests): update URL matcher test cases
- refactor(docs): add note about HTTP clients and update features list
- fix(setup.py): remove encoding param
- fix(tests): use strict equality assertion

6.16 0.1.5 / 2016-12-12

- fix(matchers): fix matching issue in URL.
- refactor(assertion): regex expression based matching must be explicitly enabled.
- feat(tests): add initial matchers tests.

6.17 0.1.4 / 2016-12-08

- refactor(README): minor changes
- fix(setup.py): lint error
- fix(#32): use explicit encoding while reading files in setup.py

6.18 0.1.3 / 2016-12-08

- fix(core): several bug fixes.
- feat(core): add pending features and major refactors.
- feat(matchers): use `unittest.TestCase` matching engine by default.

6.19 0.1.2 / 2016-12-01

- fix(matchers): runtime missing variable.

6.20 0.1.1 / 2016-12-01

- fix: Python 2 dictionary iteration syntax.
- feat(docs): add more examples.
- fix(matchers): better regular expression comparison support.

6.21 0.1.0 / 2016-11-30

- First version (still beta)

6.22 0.1.0-rc.1 / 2016-11-27

- First release candidate version (still beta)

CHAPTER 7

pook

Versatile, expressive and hackable utility library for HTTP traffic mocking and expectations made easy in Python. Heavily inspired by gock.

To get started, see the documentation, how it works, FAQ or examples.

CHAPTER 8

Features

- Simple, expressive and fluent API.
- Provides both Pythonic and chainable DSL API styles.
- Full-featured HTTP response definitions and expectations.
- Matches any HTTP protocol primitive (URL, method, query params, headers, body...).
- Full regular expressions capable mock expectations matching.
- Supports most popular HTTP clients via interceptor adapters.
- Configurable volatile, persistent or TTL limited mocks.
- Works with any testing framework/engine (unittest, pytest, nosetests...).
- First-class JSON & XML support matching and responses.
- Supports JSON Schema body matching.
- Works in both runtime and testing environments.
- Can be used as decorator and/or via context managers.
- Supports real networking mode with optional traffic filtering.
- Map/filter mocks easily for generic or custom mock expectations.
- Custom user-defined mock matcher functions.
- Simulated raised error exceptions.
- Network delay simulation (only available for `aiohttp`).
- Pluggable and hackable API.
- Customizable HTTP traffic mock interceptor engine.
- Supports third-party mocking engines, such as mocket.
- Fits good for painless test doubles.
- Does not support WebSocket traffic mocking.

- Works with Python +2.7 and +3.0 (including PyPy).
- Dependency-less: just 2 small dependencies for JSONSchema and XML tree comparison.

CHAPTER 9

Supported HTTP clients

`pook` can work with multiple mock engines, however it provides a built-in one by default, which currently supports traffic mocking in the following HTTP clients:

- urllib3 v1+
- requests v2+
- aiohttp v1+ - v2+
- urllib / http.client v2/3
- pycurl (see #16)

More HTTP clients can be supported progressively.

Note: only recent HTTP client package versions were tested.

CHAPTER 10

Installation

Using `pip` package manager (requires pip 1.8+):

```
pip install --upgrade pook
```

Or install the latest sources from Github:

```
pip install -e git+git://github.com/h2non/pook.git#egg=pook
```

CHAPTER 11

Getting started

See ReadTheDocs documentation:

CHAPTER 12

API

See annotated API reference documention.

CHAPTER 13

Examples

See examples documentation for full featured code and use case examples.

Basic mocking:

```
import pook
import requests

@pook.on
def test_my_api():
    mock = pook.get('http://twitter.com/api/1/foobar', reply=404, response_json={
↪'error': 'not found'})

    resp = requests.get('http://twitter.com/api/1/foobar')
    assert resp.status_code == 404
    assert resp.json() == {"error": "not found"}
    assert mock.calls == 1
```

Using the chainable API DSL:

```
import pook
import requests

@pook.on
def test_my_api():
    mock = (pook.get('http://twitter.com/api/1/foobar')
               .reply(404)
               .json({'error': 'not found'}))

    resp = requests.get('http://twitter.com/api/1/foobar')
    assert resp.json() == {"error": "not found"}
    assert mock.calls == 1
```

Using the decorator:

```
import pook
import requests

@pook.get('http://httpbin.org/status/500', reply=204)
@pook.get('http://httpbin.org/status/400', reply=200)
def fetch(url):
    return requests.get(url)

res = fetch('http://httpbin.org/status/400')
print('#1 status:', res.status_code)

res = fetch('http://httpbin.org/status/500')
print('#2 status:', res.status_code)
```

Simple `unittest` integration:

```
import pook
import unittest
import requests


class TestUnitTestEngine(unittest.TestCase):

    @pook.on
    def test_request(self):
        pook.get('server.com/foo').reply(204)
        res = requests.get('http://server.com/foo')
        self.assertEqual(res.status_code, 204)

    def test_request_with_context_manager(self):
        with pook.use():
            pook.get('server.com/bar', reply=204)
            res = requests.get('http://server.com/bar')
            self.assertEqual(res.status_code, 204)
```

Using the context manager for isolated HTTP traffic interception blocks:

```
import pook
import requests

# Enable HTTP traffic interceptor
with pook.use():
    pook.get('http://httpbin.org/status/500', reply=204)

    res = requests.get('http://httpbin.org/status/500')
    print('#1 status:', res.status_code)

# Interception-free HTTP traffic
res = requests.get('http://httpbin.org/status/200')
print('#2 status:', res.status_code)
```

Example using mocket Python library as underlying mock engine:

```
import pook
import requests
from mocket.plugins.pook_mock_engine import MocketEngine

```

(continues on next page)

(continued from previous page)

```
# Use mocket library as underlying mock engine
pook.set_mock_engine(MocketEngine)

# Explicitly enable pook HTTP mocking (optional)
pook.on()

# Target server URL to mock out
url = 'http://twitter.com/api/1/foobar'

# Define your mock
mock = pook.get(url,
                reply=404, times=2,
                headers={'content-type': 'application/json'},
                response_json={'error': 'foo'})

# Run first HTTP request
requests.get(url)
assert mock.calls == 1

# Run second HTTP request
res = requests.get(url)
assert mock.calls == 2

# Assert response data
assert res.status_code == 404
assert res.json() == {'error': 'foo'}

# Explicitly disable pook (optional)
pook.off()
```

Example using Hy language (Lisp dialect for Python):

```
(import [pook])
(import [requests])

(defn request [url &optional [status 404]]
  (doto (.mock pook url) (.reply status))
  (let [res (.get requests url)]
    (. res status_code)))

(defn run []
  (with [(.use pook)]
    (print "Status:" (request "http://server.com/foo" :status 204))))

;; Run test program
(defmain [&args] (run))
```

CHAPTER 14

Development

Clone the repository:

```
git clone git@github.com:h2non/pook.git
```

Install dependencies:

```
pip install -r requirements.txt -r requirements-dev.txt
```

Install Python dependencies:

```
make install
```

Lint code:

```
make lint
```

Run tests:

```
make test
```

Generate documentation:

```
make htmldocs
```

CHAPTER 15

License

MIT - Tomas Aparicio

CHAPTER 16

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

U

X