# clients Documentation

*Release 1.2*

**Aric Coady**

**Jan 10, 2020**

# Contents

HTTP for humanitarians.

# CHAPTER 1

# Quickstart

As great as requests is, typical usage is falling into some anti-patterns.

- Being url-based, realistically all code needs to deal with url joining. Which tends to be redundant and suffer from leading or trailing slash issues.
- The module level methods don't take advantage of connection pooling, and require duplicate settings. Given the "100% automatic" documentation of connection reuse, it's unclear how widely known this is.
- Using Sessions requires assigning every setting individually, and still requires url joining.

Clients aim to be encourage best practices by making Sessions even easier to use than the module methods. Examples use the httpbin client testing service.

```
client = clients.Client(url, auth=('user', 'pass'), headers={'x-test': 'true'})
r = client.get('headers', headers={'x-test2': 'true'})
assert {'x-test', 'x-test2'} <= set(r.request.headers)

r = client.get('cookies', cookies={'from-my': 'browser'})
assert r.json() == {'cookies': {'from-my': 'browser'}}
r = client.get('cookies')
assert r.json() == {'cookies': {}}

client.get('cookies/set', params={'sessioncookie': '123456789'})
r = client.get('cookies')
assert r.json() == {'cookies': {'sessioncookie': '123456789'}}
```

Which reveals another anti-pattern regarding Responses. Although the response object is sometimes required, naturally the most common use case is to access the content. But the onus is on the caller to check the `status_code` and `content-type`.

Resources aim to making writing custom api clients or sdks easier. Their primary feature is to allow direct content access without silencing errors. Response content type is inferred from headers: `json`, `content`, or `text`.

```
resource = clients.Resource(url)
assert resource.get('get')['url'] == url + '/get'
```

(continues on next page)

(continued from previous page)

```
with pytest.raises(IOError):
    resource.get('status/404')
assert '<html>' in resource.get('html')
assert isinstance(resource.get('bytes/10'), bytes)
```

CHAPTER 2

# Advanced Usage

`Clients` allow any base url, not just hosts, and consequently support path concatenation. Following the semantics of `urljoin` however, absolute paths and urls are treated as such. Hence there's no need to parse a url retrieved from an api.

```
client = clients.Client(url)
cookies = client / 'cookies'
assert isinstance(cookies, clients.Client)
assert cookies.get().url == url + '/cookies'

assert cookies.get('/').url == url + '/'
assert cookies.get(url).url == url + '/'
```

Some api endpoints require trailing slashes; some forbid them. Set it and forget it.

```
client = clients.Client(url, trailing='/')
assert client.get('ip').status_code == 404
```

Note `trailing` isn't limited to only being a slash. This can be useful for static paths below a parameter: `api/v1/{query}.json`.

CHAPTER 3

# Asyncio

Using httpx instead of requests, AsyncClients and AsyncResources implement the same interface, except the request methods return asyncio coroutines.

# CHAPTER 4

## Avant-garde Usage

`Resources` support operator overloaded syntax wherever sensible. These interfaces often obviate the need for writing custom clients specific to an API.

- `__getattr__`: alternate path concatenation
- `__getitem__`: GET content
- `__setitem__`: PUT json
- `__delitem__`: DELETE
- `__contains__`: HEAD ok
- `__iter__`: GET streamed lines or content
- `__call__`: GET with params

```
resource = clients.Resource(url)
assert set(resource['get']) == {'origin', 'headers', 'args', 'url'}
resource['put'] = {}
del resource['delete']

assert '200' in resource.status
assert '404' not in resource.status
assert [line['id'] for line in resource / 'stream/3'] == [0, 1, 2]
assert next(iter(resource / 'html')) == '<!DOCTYPE html>'
assert resource('cookies/set', name='value') == {'cookies': {'name': 'value'}}
```

Higher-level methods for common requests.

- `iter`: __iter__ with args
- `update`: PATCH with json params, or GET with conditional PUT
- `create`: POST and return location
- `download`: GET streamed content to file
- `authorize`: acquire oauth token

```
resource = clients.Resource(url)
assert list(map(len, resource.iter('stream-bytes/256'))) == [128] * 2
assert resource.update('patch', name='value')['json'] == {'name': 'value'}
assert resource.create('post', {'name': 'value'}) is None
file = resource.download(io.BytesIO(), 'image/png')
assert file.tell()
```

A singleton decorator can be used on subclasses, conveniently creating a single custom instance.

```
@clients.singleton('http://localhost/')
class custom_api(clients.Resource):
    pass  # custom methods

assert isinstance(custom_api, clients.Resource)
assert custom_api.url == 'http://localhost/'
```

Remote and AsyncRemote clients default to POSTs with json bodies, for APIs which are more RPC than REST.

Graph and AsyncGraph remote clients execute GraphQL queries.

Proxy and AsyncProxy clients provide load-balancing across multiple hosts, with an extensible interface for different algorithms.

# CHAPTER 5

Contents:

## 5.1 Client

**class** clients.**Client**(*url*, *trailing=''*, *headers=()*, *auth=None*, ***attrs*)

Bases: requests.sessions.Session

A Session which sends requests to a base url.

**Parameters**

- **url** – base url for requests
- **trailing** – trailing chars (e.g. /) appended to the url
- **headers** – additional headers to include in requests
- **auth** – additional authorization support for {token_type: access_token}, available per request as well
- **attrs** – additional Session attributes

**__truediv__**(*path: str*) → clients.base.Client

Return a cloned client with appended path.

**delete**(*path=''*, ***kwargs*)

DELETE request with optional path.

**get**(*path=''*, ***kwargs*)

GET request with optional path.

**head**(*path=''*, *allow_redirects=False*, ***kwargs*)

HEAD request with optional path.

**options**(*path=''*, ***kwargs*)

OPTIONS request with optional path.

**patch**(*path=''*, *json=None*, ***kwargs*)

PATCH request with optional path and json body.

**post** (*path=''*, *json=None*, ***kwargs*)
POST request with optional path and json body.

**put** (*path=''*, *json=None*, ***kwargs*)
PUT request with optional path and json body.

**request** (*method*, *path*, *auth=None*, ***kwargs*)
Send request with relative or absolute path and return response.

## 5.2 Resource

**class** clients.**Resource** (*url*, *trailing=''*, *headers=()*, *auth=None*, ***attrs*)
Bases: clients.base.Client

A *Client* which returns json content and has syntactic support for requests.

**content_type** (*response*)
Return name {*json*, *text*,... } of response's content_type.

**__call__** (*path: str = ''*, ***params*)
GET request with params.

**__contains__** (*path: str*)
Return whether endpoint exists according to HEAD request.

**__delitem__** (*path=''*, ***kwargs*)
DELETE request with optional path.

**__getattr__** (*name: str*) → clients.base.Client
Return a cloned client with appended path.

**__getitem__** (*path=''*, ***kwargs*)
GET request with optional path.

**__iter__** (*path: str = ''*, ***kwargs*) → Iterator[T_co]
Iterate lines or chunks from streamed GET request.

**__setitem__** (*path=''*, *json=None*, ***kwargs*)
PUT request with optional path and json body.

**authorize** (*path: str = ''*, ***kwargs*) → dict
Acquire oauth access token and set `auth`.

**client**
upcasted *Client*

**content_type = functools.partial(<function content_type>, text='text/', json='applicat**

**create** (*path: str = ''*, *json=None*, ***kwargs*) → str
POST request and return location.

**download** (*file*, *path: str = ''*, ***kwargs*)
Output streamed GET request to file.

**iter** (*path: str = ''*, ***kwargs*) → Iterator[T_co]
Iterate lines or chunks from streamed GET request.

**request** (*method*, *path*, ***kwargs*)
Send request with path and return processed content.

**update** (*path: str = ''*, *callback: Callable = None*, ***json*)
PATCH request with json params.

**Parameters** **callback** – optionally update with GET and validated PUT. `callback` is called on the json result with keyword params, i.e., `dict` correctly implements the simple update case.

**updating**(*path: str = ''*, ***kwargs*)
Provisional context manager to GET and conditionally PUT json data.

## 5.3 Remote

**class** clients.**Remote**(*url: str*, *json=()*, ***kwargs*)
Bases: `clients.base.Client`

A *Client* which defaults to posts with json bodies, i.e., RPC.

**Parameters**

- **url** – base url for requests
- **json** – default json body for all calls
- **kwargs** – same options as *Client*

**__call__**(*path: str = ''*, ***json*)
POST request with json body and *check()* result.

**__getattr__**(*name: str*) → clients.base.Client
Return a cloned client with appended path.

**__init__**(*url: str*, *json=()*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

**static check**(*result*)
Override to return result or raise error, for APIs which don't use status codes.

**client**
upcasted *Client*

## 5.4 Graph

**class** clients.**Graph**(*url: str*, *json=()*, ***kwargs*)
Bases: `clients.base.Remote`

A *Remote* client which executes GraphQL queries.

**Error**
alias of `requests.exceptions.HTTPError`

**classmethod check**(*result: dict*)
Return `data` or raise `errors`.

**execute**(*query: str*, ***variables*)
Execute query over POST.

## 5.5 Proxy

**class** clients.**Proxy**(**urls*, ***kwargs*)
Bases: `clients.base.Client`

An extensible embedded proxy client to multiple hosts.

The default implementation provides load balancing based on active connections. It does not provide error handling or retrying.

**Parameters**

- **urls** – base urls for requests
- **kwargs** – same options as *Client*

**class Stats**
Bases: collections.Counter

Thread-safe Counter.

Context manager tracks number of active connections and errors.

**add**(***kwargs*)
Atomically add data.

**choice**(*method: str*) → str
Return chosen url according to priority.

**Parameters** **method** – placeholder for extensions which distinguish read/write requests

**priority**(*url: str*)
Return comparable priority for url.

Minimizes errors, failures (500s), and active connections. None may be used to eliminate from consideration.

**request**(*method*, *path*, ***kwargs*)
Send request with relative or absolute path and return response.

## 5.6 AsyncClient

**class** clients.**AsyncClient**(*url: str*, *, *trailing: str = ''*, *auth=None*, ***attrs*)
Bases: httpx.client.AsyncClient

An asynchronous Client which sends requests to a base url.

**Parameters**

- **url** – base url for requests
- **trailing** – trailing chars (e.g. */*) appended to the url
- **params** – default query params
- **auth** – additional authorization support for {token_type: access_token}, available per request as well
- **attrs** – additional AsyncClient options

**__truediv__**(*path: str*) → clients.base.Client
Return a cloned client with appended path.

**delete**(*path=''*, ***kwargs*)
DELETE request with optional path.

**get**(*path=''*, ***kwargs*)
GET request with optional path.

**head** (*path=''*, *allow_redirects=False*, ***kwargs*)
HEAD request with optional path.

**options** (*path=''*, ***kwargs*)
OPTIONS request with optional path.

**patch** (*path=''*, *json=None*, ***kwargs*)
PATCH request with optional path and json body.

**post** (*path=''*, *json=None*, ***kwargs*)
POST request with optional path and json body.

**put** (*path=''*, *json=None*, ***kwargs*)
PUT request with optional path and json body.

**request** (*method*, *path*, *auth=None*, ***kwargs*)
Send request with relative or absolute path and return response.

**run** (*name: str*, **args*, ***kwargs*)
Synchronously call method and run coroutine.

## 5.7 AsyncResource

**class** clients.**AsyncResource** (*url: str*, ***, *trailing: str = ''*, *auth=None*, ***attrs*)
Bases: clients.aio.AsyncClient

An *AsyncClient* which returns json content and has syntactic support for requests.

**__call__** (*path: str = ''*, ***params*)
GET request with params.

**__getattr__** (*path: str*) → clients.base.Client
Return a cloned client with appended path.

**__getitem__** (*path=''*, ***kwargs*)
GET request with optional path.

**authorize** (*path: str = ''*, ***kwargs*) → dict
Acquire oauth access token and set auth.

**client**
upcasted *AsyncClient*

**content_type = functools.partial(<function content_type>, text='text/', json='applicat**

**request** (*method*, *path*, ***kwargs*)
Send request with path and return processed content.

**update** (*path=''*, *callback=None*, ***json*)
PATCH request with json params.

**Parameters** **callback** – optionally update with GET and validated PUT. callback is called on the json result with keyword params, i.e., dict correctly implements the simple update case.

**updating** (*path: str = ''*, ***kwargs*)
Provisional context manager to GET and conditionally PUT json data.

## 5.8 AsyncRemote

**class** clients.**AsyncRemote**(*url: str*, *json=()*, ***kwargs*)
Bases: clients.aio.AsyncClient

An *AsyncClient* which defaults to posts with json bodies, i.e., RPC.

**Parameters**

- **url** – base url for requests
- **json** – default json body for all calls
- **kwargs** – same options as *AsyncClient*

**__call__**(*path=''*, ***json*)
POST request with json body and check result.

**__getattr__**(*path: str*) → clients.base.Client
Return a cloned client with appended path.

**__init__**(*url: str*, *json=()*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

**static check**(*result*)
Override to return result or raise error, for APIs which don't use status codes.

**client**
upcasted *AsyncClient*

## 5.9 AsyncGraph

**class** clients.**AsyncGraph**(*url: str*, *json=()*, ***kwargs*)
Bases: clients.aio.AsyncRemote

An *AsyncRemote* client which executes GraphQL queries.

**Error**
alias of httpx.exceptions.HTTPError

**classmethod check**(*result: dict*)
Return data or raise errors.

**execute**(*query: str*, ***variables*)
Execute query over POST.

## 5.10 AsyncProxy

**class** clients.**AsyncProxy**(**urls*, ***kwargs*)
Bases: clients.aio.AsyncClient

An extensible embedded proxy client to multiple hosts.

The default implementation provides load balancing based on active connections. It does not provide error handling or retrying.

**Parameters**

- **urls** – base urls for requests

- **kwargs** – same options as *AsyncClient*

**class Stats**
Bases: collections.Counter

Thread-safe Counter.

Context manager tracks number of active connections and errors.

**add**(***kwargs*)
Atomically add data.

**choice**(*method: str*) → str
Return chosen url according to priority.

**Parameters** **method** – placeholder for extensions which distinguish read/write requests

**priority**(*url: str*)
Return comparable priority for url.

Minimizes errors, failures (500s), and active connections. None may be used to eliminate from consideration.

**request**(*method*, *path*, ***kwargs*)
Send request with relative or absolute path and return response.

## 5.11 singleton

clients.**singleton**(**args*, ***kwargs*)
Return a decorator for singleton class instances.

CHAPTER 6

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## C

# Index

## Symbols



## A



## C



## D



## E



## G



## H



## I



## O



## P

## R



## S



## U