CONTENTS

1 Why? 3

2 Quick Start 5
  2.1 GET ................................. 5
  2.2 POST ............................... 5

3 Why not 100% requests-alike? 7

4 Feature Parity with Requests 9

5 Table of Contents 11
  5.1 Use Cases ............................ 11
      5.1.1 Handling Streaming Responses 11
      5.1.2 URLs, URIs, and Hyperlinks 11
      5.1.3 Query Parameters 12
      5.1.4 JSON .......................... 13
      5.1.5 Auth ............................ 13
      5.1.6 Redirects ....................... 13
      5.1.7 Cookies ......................... 14
      5.1.8 Customizing the Twisted Agent 15
  5.2 Testing Helpers ..................... 15
      5.2.1 Writing tests for HTTP clients 15
      5.2.2 Writing tests for Twisted Web resources 17
  5.3 API Reference ...................... 17
      5.3.1 Making Requests .................. 18
      5.3.2 Accessing Content ............... 20
  5.4 The HTTP Client .................... 20
      5.4.1 Augmented Response Objects ....... 21
      5.4.2 Authentication .................... 22
      5.4.3 Test Helpers ...................... 23
      5.4.4 MultiPartProducer Objects ...... 26
  5.5 Changelog ........................... 27
      5.5.1 22.1.0 (2022-01-29) ............... 27
      5.5.2 21.5.0 (2021-05-24) ............... 27
      5.5.3 21.1.0 (2021-01-14) ............... 27
      5.5.4 20.9.0 (2020-09-27) ............... 28
      5.5.5 20.4.1 (2020-04-16) ............... 29
      5.5.6 20.4.0 (2020-04-16) ............... 29
      5.5.7 20.3.0 (2020-03-15) ............... 29

6 Indices and tables 31
Release v22.1.0 (What’s new?).

treq depends on a recent Twisted and functions on Python 2.7 and Python 3.3+ (including PyPy).
requests by Kenneth Reitz is a wonderful library. I want the same ease of use when writing Twisted applications. treq is not of course a perfect clone of requests. I have tried to stay true to the do-what-I-mean spirit of the requests API and also kept the API familiar to users of Twisted and twisted.web.client.Agent on which treq is based.
Installation

$ pip install treq

## 2.1 GET

```python
def main(reactor, *args):
    d = treq.get('https://httpbin.org/get')
    d.addCallback(print_response)
    return d
```

Full example: basic_get.py

## 2.2 POST

```python
def main(reactor):
    d = treq.post("https://httpbin.org/post",
                  data={"form": "data"})
    d.addCallback(print_response)
    return d
```

Full example: basic_post.py
WHY NOT 100% REQUESTS-ALIKE?

Initially when I started off working on treq I thought the API should look exactly like requests except anything that would involve the network would return a Deferred.

Over time while attempting to mimic the requests API it became clear that not enough code could be shared between requests and treq for it to be worth the effort to translate many of the usage patterns from requests.

With the current version of treq I have tried to keep the API simple, yet remain familiar to users of Twisted and its lower-level HTTP libraries.
Even though mimicking the requests API is not a goal, supporting most of its features is. Here is a list of requests features and their status in treq.
5.1 Use Cases

5.1.1 Handling Streaming Responses

In addition to receiving responses with IResponse.deliverBody(), treq provides a helper function treq.collect() which takes a response and a single argument function which will be called with all new data available from the response. Much like IProtocol.dataReceived(), treq.collect() knows nothing about the framing of your data and will simply call your collector function with any data that is currently available.

Here is an example which simply a file object’s write method to treq.collect() to save the response body to a file.

```python
def download_file(reactor, url, destination_filename):
    destination = open(destination_filename, 'wb')
    d = treq.get(url, unbuffered=True)
    d.addCallback(treq.collect, destination.write)
    d.addBoth(lambda _: destination.close())
    return d
```

Full example: download_file.py

5.1.2 URLs, URIs, and Hyperlinks

The url argument to HTTPClient.request() accepts three URL representations:

- High-level: hyperlink.DecodedURL
- Mid-level str (unicode on Python 2)
- Low-level: ASCII bytes or hyperlink.URL

The high-level DecodedURL form is useful when programatically generating URLs. Here is an example that builds a URL that contains a & character, which is automatically escaped properly.

```python
def main(reactor):
    url = (DecodedURL.from_text(u"https://httpbin.org")
          .child(u"get")  # add path /get
          .add(u"foo", u"&")  # add query ?foo=%26
          )
    print(url.to_text())
    return treq.get(url).addCallback(print_response)
```
Full example: basic_url.py

5.1.3 Query Parameters

treq.HTTPClient.request() supports a params keyword argument which will be URL-encoded and added to the url argument in addition to any query parameters that may already exist.

The params argument may be either a dict or a list of (key, value) tuples.

If it is a dict then the values in the dict may either be scalar values or a list or tuple thereof. Scalar values means str, bytes, or anything else — even None — which will be coerced to str. Strings are UTF-8 encoded.

```python
@inlineCallbacks
def main(reactor):
    print('List of tuples')
    resp = yield treq.get('https://httpbin.org/get',
                          params=[('foo', 'bar'), ('baz', 'bax')])
    content = yield resp.text()
    print(content)

    print('Single value dictionary')
    resp = yield treq.get('https://httpbin.org/get',
                          params={'foo': 'bar', 'baz': 'bax'})
    content = yield resp.text()
    print(content)

    print('Multi value dictionary')
    resp = yield treq.get('https://httpbin.org/get',
                          params={b'foo': [b'bar', b'baz', b'bax']})
    content = yield resp.text()
    print(content)

    print('Mixed value dictionary')
    resp = yield treq.get('https://httpbin.org/get',
                          params={'foo': [1, 2, 3], 'bax': b'quux', b'bar': 'foo'})
    content = yield resp.text()
    print(content)

    print('Preserved query parameters')
    resp = yield treq.get('https://httpbin.org/get?foo=bar',
                          params={'baz': 'bax'})
    content = yield resp.text()
    print(content)
```

Full example: query_params.py

If you prefer a strictly-typed API, try hyperlink.DecodedURL. Use its add() and set() methods to add query parameters without risk of accidental type coercion.
5.1.4 JSON

`HTTPClient.request()` supports a `json` keyword argument that gives a data structure to serialize as JSON (using `json.dumps()`). This also implies a `Content-Type: application/json` request header. The `json` parameter is mutually-exclusive with `data`.

The `_Response.json()` method decodes a JSON response body. It buffers the whole response and decodes it with `json.loads()`.

```python
@defer.inlineCallbacks
def main(reactor):
    response = yield treq.post(
        'https://httpbin.org/post',
        json={'msg': 'Hello!'},
    )
    data = yield response.json()
    pprint(data)
```

Full example: `json_post.py`

5.1.5 Auth

HTTP Basic authentication as specified in RFC 2617 is easily supported by passing an `auth` keyword argument to any of the request functions.

The `auth` argument should be a tuple of the form ('username', 'password').

```python
def main(reactor, *args):
    d = treq.get(
        'https://httpbin.org/basic-auth/treq/treq',
        auth=('treq', 'treq')
    )
    d.addCallback(print_response)
    return d
react(main, [])
```

Full example: `basic_auth.py`

5.1.6 Redirects

treq handles redirects by default.

The following will print a 200 OK response.

```python
def main(reactor, *args):
    d = treq.get('https://httpbin.org/redirect/1')
    d.addCallback(print_response)
    return d
react(main, [])
```

Full example: `redirects.py`

You can easily disable redirects by simply passing `allow_redirects=False` to any of the request methods.

5.1. Use Cases
```python
def main(reactor, *args):
    d = treq.get('https://httpbin.org/redirect/1', allow_redirects=False)
    d.addCallback(print_response)
    return d

react(main, [])
```

Full example: disable_redirects.py

You can even access the complete history of treq response objects by calling the history() method on the response.

```python
def main(reactor, *args):
    d = treq.get('https://httpbin.org/redirect/1')

    def cb(response):
        print('Response history: ')
        print(response.history())
        return print_response(response)

    d.addCallback(cb)

Full example: response_history.py

### 5.1.7 Cookies

Cookies can be set by passing a dict or cookielib.CookieJar instance via the cookies keyword argument. Later cookies set by the server can be retrieved using the cookies() method of the response.

The object returned by cookies() supports the same key/value access as requests cookies.

```python
def main(reactor, *args):
    d = treq.get('https://httpbin.org/cookies/set?hello=world')

    def _get_jar(resp):
        jar = resp.cookies()

        print('The server set our hello cookie to: {}').format(jar['hello']))

        return treq.get('https://httpbin.org/cookies', cookies=jar)

    d.addCallback(_get_jar)
    d.addCallback(print_response)

    return d

Full example: using_cookies.py
```
5.1.8 Customizing the Twisted Agent

The main `treq` module has helper functions that automatically instantiate an instance of `treq.client.HTTPClient`. You can create an instance of `HTTPClient` directly in order to customize the parameters used to initialize it. Internally, the `HTTPClient` wraps an instance of `twisted.web.client.Agent`. When you create an instance of `HTTPClient`, you must initialize it with an instance of `Agent`. This allows you to customize its behavior.

```python
def make_custom_agent(reactor):
    return Agent(reactor, connectTimeout=42)

def main(reactor, *args):
    agent = make_custom_agent(reactor)
    http_client = HTTPClient(agent)
    d = http_client.get(
        'https://secure.example.net/area51',
        auth=('admin', 'you\'ll never guess!'))
    d.addCallback(print_response)
    return d
```

Full example: `custom_agent.py`

5.2 Testing Helpers

The `treq.testing` module provides some tools for testing both HTTP clients which use the treq API and implementations of the Twisted Web resource model.

5.2.1 Writing tests for HTTP clients

The `StubTreq` class implements the `treq` module interface (`treq.get()`, `treq.post()`, etc.) but runs all I/O via a `MemoryReactor`. It wraps a `twisted.web.resource.IResource` provider which handles each request.

You can wrap a pre-existing `IResource` provider, or write your own. For example, the `twisted.web.resource.ErrorPage` resource can produce an arbitrary HTTP status code. `twisted.web.static.File` can serve files or directories. And you can easily achieve custom responses by writing trivial resources yourself:

```python
@implementer(IResource)
class JsonResource(object):
    isLeaf = True  # NB: means getChildWithDefault will not be called

    def __init__(self, data):
        self.data = data

    def render(self, request):
        request.setHeader(b'Content-Type', b'application/json')
        return json.dumps(self.data).encode('utf-8')
```

However, those resources don’t assert anything about the request. The `RequestSequence` and `StringStubbingResource` classes make it easy to construct a resource which encodes the expected request...
and response pairs. Do note that most parameters to these functions must be bytes—it’s safest to use the `b''` string syntax, which works on both Python 2 and 3.

For example:

```python
from twisted.internet import defer
from twisted.trial.unittest import SynchronousTestCase
from twisted.web import http

from treq.testing import StubTreq, HasHeaders
from treq.testing import RequestSequence, StringStubbingResource

@defer.inlineCallbacks
def make_a_request(treq):
    """Make a request using treq."
    response = yield treq.get('http://an.example/foo',
                               params={'a': 'b'},
                               headers={'Accept': 'application/json'})
    if response.code == http.OK:
        result = yield response.json()
    else:
        message = yield response.text()
        raise Exception("Got an error from the server: {}".format(message))
    defer.returnValue(result)

class MakeARequestTests(SynchronousTestCase):
    """Test :func:`make_a_request()` using :mod:`treq.testing.RequestSequence`.
    ""

def test_200_ok(self):
    """On a 200 response, return the response's JSON.""
    req_seq = RequestSequence([((b'get', 'http://an.example/foo', {b'a': [b'b']}),
                                HasHeaders({'Accept': ['application/json']}, b''),
                                (http.OK, {b'Content-Type': 'application/json'}, b'"status": "ok"'))])
    treq = StubTreq(StringStubbingResource(req_seq))
    with req_seq.consume(self.fail):
        result = self.successResultOf(make_a_request(treq))
        self.assertEqual({"status": "ok"}, result)

def test_418_teapot(self):
    """On an unexpected response code, raise an exception""
    req_seq = RequestSequence([((b'get', 'http://an.example/foo', {b'a': [b'b']}),
                                HasHeaders({'Accept': ['application/json']}, b''),
                                (418, {b'Content-Type': 'text/plain'}, b'I'm a teapot!'))])
```

(continues on next page)
treq = StubTreq(StringStubbingResource(req_seq))

```python
with req_seq.consume(self.fail):
    failure = self.failureResultOf(make_a_request(treq))

self.assertEqual(u"Got an error from the server: I'm a teapot!", failure.getErrorMessage())
```

This may be run with trial testing_seq.py. Download: testing_seq.py.

**Loosely matching the request**

If you don’t care about certain parts of the request, you can pass `unittest.mock.ANY`, which compares equal to anything. This sequence matches a single GET request with any parameters or headers:

```python
from unittest.mock import ANY

RequestSequence([
    ((b'get', ANY, ANY, b''), (200, {}, b'ok'))
])
```

If you care about headers, use `HasHeaders` to make assertions about the headers present in the request. It compares equal to a superset of the headers specified, which helps make your test robust to changes in treq or Agent. Right now treq adds the Accept-Encoding: gzip header, but as support for additional compression methods is added, this may change.

**5.2.2 Writing tests for Twisted Web resources**

Since `StubTreq` wraps any resource, you can use it to test your server-side code as well. This is superior to calling your resource’s methods directly or passing mock objects, since it uses a real `Agent` to generate the request and a real `Site` to process the response. Thus, the `request` object your code interacts with is a real `twisted.web.server.Request` and behaves the same as it would in production.

Note that if your resource returns `NOT_DONE_YET` you must keep a reference to the `RequestTraversalAgent` and call its `flush()` method to spin the memory reactor once the server writes additional data before the client will receive it.

**5.3 API Reference**

This page lists all of the interfaces exposed by the `treq` package.
5.3.1 Making Requests

The treq module provides several convenience functions for making requests. These functions all create a default treq.client.HTTPClient instance and pass their arguments to the appropriate HTTPClient method.

treq.request(method, url, **kwargs)
Make an HTTP request.

Parameters

- **method** *(str)* – HTTP method. Example: 'GET', 'HEAD', 'PUT', 'POST'.
- **url** *(hyperlink.DecodedURL, str, bytes, or hyperlink.EncodedURL)* – http or https URL, which may include query arguments.
- **headers** *(Headers or None)* – Optional HTTP Headers to send with this request.
- **params** *(dict w/ str or list/tuple of str values, list of 2-tuples, or None.)* – Optional parameters to be append to the URL query string. Any query string parameters in the url will be preserved.
- **data** *(bytes, typing.BinaryIO, IBodyProducer, or None)* – Arbitrary request body data.

If **files** is also passed this must be a dict, a tuple or list of field tuples as accepted by MultiPartProducer. The request is assigned a Content-Type of multipart/form-data.

If a dict, list, or tuple it is URL-encoded and the request assigned a Content-Type of application/x-www-form-urlencoded.

Otherwise, any non-None value is passed to the client’s data_to_body_producer callable (by default, IBodyProducer), which accepts bytes and binary files like returned by open(.. , "rb").

- **files** – Files to include in the request body, in any of the several formats:
  - ["fieldname", binary_file]
  - ["fieldname", "filename", binary_file]
  - ["fieldname", "filename", "content-type", binary_file]

Or a mapping:
  - {"fieldname": binary_file}
  - {"fieldname": ("filename", binary_file)}
  - {"fieldname": ("filename", "content-type", binary_file)}

Each binary_file is a file-like object open in binary mode (like returned by open("filename", "rb")). The filename is taken from the file’s name attribute if not specified. The Content-Type is guessed based on the filename using mimetypes.guess_type() if not specified, falling back to application/octet-stream.

While uploading Treq will measure the length of seekable files to populate the Content-Length header of the file part.

If **files** is given the request is assigned a Content-Type of multipart/form-data. Additional fields may be given in the data argument.

- **json** *(dict, list, tuple, int, str, bool, or None)* – Optional JSON-serializable content for the request body. Mutually exclusive with data and files.

- **auth** *(tuple of ('username', 'password'))* – HTTP Basic Authentication information — see treq.auth.add_auth().
• **cookies** (dict or cookielib.CookieJar) – Cookies to send with this request. The HTTP kind, not the tasty kind.

• **timeout** (int) – Request timeout seconds. If a response is not received within this time-frame, a connection is aborted with CancelledError.

• **allow_redirects** (bool) – Follow HTTP redirects. Default: True

• **browser_like_redirects** (bool) – Follow redirects like a web browser: When a 301 or 302 redirect is received in response to a POST request convert the method to GET. See 7231 and BrowserLikeRedirectAgent). Default: False

• **unbuffered** (bool) – Pass True to to disable response buffering. By default treq buffers the entire response body in memory.

• **reactor** – Optional Twisted reactor.

• **persistent** (bool) – Use persistent HTTP connections. Default: True

• **agent** (twisted.web.iweb.IAgent) – Provide your own custom agent. Use this to override things like connectTimeout or BrowserLikePolicyForHTTPS. By default, treq will create its own Agent with reasonable defaults.

**Return type** Deferred that fires with an IResponse

Changed in version treq: 20.9.0

The **url** param now accepts hyperlink.DecodedURL and hyperlink.EncodedURL objects.

treq.get-url, headers=None, **kwargs
Make a GET request.

See treq.request()

treq.head-url, **kwargs
Make a HEAD request.

See treq.request()

treq.post-url, data=None, **kwargs
Make a POST request.

See treq.request()

treq.put-url, data=None, **kwargs
Make a PUT request.

See treq.request()

treq.patch-url, data=None, **kwargs
Make a PATCH request.

See treq.request()

treq.delete-url, **kwargs
Make a DELETE request.

See treq.request()
5.3.2 Accessing Content

treq.collect(response, collector)
Incrementally collect the body of the response.

This function may only be called once for a given response.

Parameters
- response (IResponse) – The HTTP response to collect the body from.
- collector (single argument callable) – A callable to be called each time data is available from the response body.

Return type Deferred that fires with None when the entire body has been read.

treq.content(response)
Read the contents of an HTTP response.

This function may be called multiple times for a response, it uses a WeakKeyDictionary to cache the contents of the response.

Parameters response (IResponse) – The HTTP Response to get the contents of.

Return type Deferred that fires with the content as a str.

treq.text_content(response, encoding='ISO-8859-1')
Read the contents of an HTTP response and decode it with an appropriate charset, which may be guessed from the Content-Type header.

Parameters
- response (IResponse) – The HTTP Response to get the contents of.
- encoding (str) – A charset, such as UTF-8 or ISO-8859-1, used if the response does not specify an encoding.

Return type Deferred that fires with a unicode string.

treq.json_content(response, **kwargs)
Read the contents of an HTTP response and attempt to decode it as JSON.

This function relies on content() and so may be called more than once for a given response.

Parameters
- response (IResponse) – The HTTP Response to get the contents of.
- kwargs – Any keyword arguments accepted by json.loads()

Return type Deferred that fires with the decoded JSON.

5.4 The HTTP Client

treq.client.HTTPClient has methods that match the signatures of the convenience request functions in the treq module.

class treq.client.HTTPClient(agent, cookiejar=None, data_to_body_producer=IBodyProducer)
request(method, url, *, params=None, headers=None, data=None, files=None, json=<object object>, auth=None, cookies=None, allow_redirects=True, browser_like_redirects=False, unbuffered=False, reactor=None, timeout=None, _stacklevel=2)

See treq.request().
get(
  url, **kwargs
)
See treq.get().

head(
  url, **kwargs
)
See treq.head().

pst(
  url, **kwargs
)
See treq.post().

put(
  url, **kwargs
)
See treq.put().

patch(
  url, **kwargs
)
See treq.patch().

delete(
  url, **kwargs
)
See treq.delete().

5.4.1 Augmented Response Objects

treq.request(), treq.get(), etc. return an object which provides twisted.web.iweb.IResponse, plus a few additional convenience methods:

class treq.response._Response

collect(collector)

Incrementally collect the body of the response, per treq.collect().

Parameters collector – A single argument callable that will be called with chunks of body data as it is received.

Returns A Deferred that fires when the entire body has been received.

content()

Read the entire body all at once, per treq.content().

Returns A Deferred that fires with a bytes object when the entire body has been received.

json(**kwargs)

Collect the response body as JSON per treq.json_content().

Parameters kwargs – Any keyword arguments accepted by json.loads()

Return type Deferred that fires with the decoded JSON when the entire body has been read.

text(encoding='ISO-8859-1')

Read the entire body all at once as text, per treq.text_content().

Return type A Deferred that fires with a unicode string when the entire body has been received.

history()

Get a list of all responses that (such as intermediate redirects), that ultimately ended in the current response. The responses are ordered chronologically.

Returns A list of _Response objects

cookies()

Get a copy of this response’s cookies.

Return type requests.cookies.RequestsCookieJar

Inherited from twisted.web.iweb.IResponse:

Variables

5.4. The HTTP Client
treq Documentation, Release 22.1.0

- `version` – See `IResponse.version`
- `code` – See `IResponse.code`
- `phrase` – See `IResponse.phrase`
- `headers` – See `IResponse.headers`
- `length` – See `IResponse.length`
- `request` – See `IResponse.request`
- `previousResponse` – See `IResponse.previousResponse`

`deliverBody(protocol)`
See `IResponse.deliverBody()`

`setPreviousResponse(response)`
See `IResponse.setPreviousResponse()`

### 5.4.2 Authentication

`treq.auth.add_auth(agent, auth_config)`
Wrap an agent to perform authentication

**Parameters**
- `agent` – Agent to wrap.
- `auth_config` – A (`username`, `'password'`) tuple — see `add_basic_auth()`.

**Returns** `IAgent`

**Raises** `UnknownAuthConfig` – When the format `auth_config` isn’t supported.

`treq.auth.add_basic_auth(agent: <InterfaceClass twisted.web.iweb.IAgent>, username: typing.Union[str, bytes], password: typing.Union[str, bytes]) → <InterfaceClass twisted.web.iweb.IAgent>`
Wrap an agent to add HTTP basic authentication

The returned agent sets the `Authorization` request header according to the basic authentication scheme described in RFC 7617. This header contains the given `username` and `password` in plaintext, and thus should only be used over an encrypted transport (HTTPS).

Note that the colon (:) is used as a delimiter between the `username` and `password`, so if either parameter includes a colon the interpretation of the `Authorization` header is server-defined.

**Parameters**
- `agent` – Agent to wrap.
- `username` – The username.
- `password` – The password.

**Returns** `IAgent`

`exception treq.auth.UnknownAuthConfig(config)`
The authentication config provided couldn’t be interpreted.
5.4.3 Test Helpers

The `treq.testing` module contains tools for in-memory testing of HTTP clients and servers.

**StubTreq Objects**

class `treq.testing.StubTreq(resource)`

StubTreq implements the same interface as the `treq` module or the `HTTPClient` class, with the limitation that it does not support the `files` argument.

flush()

Flush all data between pending client/server pairs.

This is only necessary if a `Resource` under test returns `NOT_DONE_YET` from its `render` method, making a response asynchronous. In that case, after each write from the server, `flush()` must be called so the client can see it.

As the methods on `treq.client.HTTPClient`:

request()

See `treq.request()`.

get()

See `treq.get()`.

head()

See `treq.head()`.

post()

See `treq.post()`.

put()

See `treq.put()`.

patch()

See `treq.patch()`.

delete()

See `treq.delete()`.

**RequestTraversalAgent Objects**

class `treq.testing.RequestTraversalAgent(rootResource)`

IAgent implementation that issues an in-memory request rather than going out to a real network socket.

flush()

Flush all data between pending client/server pairs.

This is only necessary if a `Resource` under test returns `NOT_DONE_YET` from its `render` method, making a response asynchronous. In that case, after each write from the server, `flush()` must be called so the client can see it.

request(`method`, `uri`, `headers=None`, `bodyProducer=None`)

Implement IAgent.request.
RequestSequence Objects

class treq.testing.RequestSequence(sequence, async_failure_reporter=None)

For an example usage, see RequestSequence.consume().

Takes a sequence of:

```python
[((method, url, params, headers, data), (code, headers, body)), ...
```

Expects the requests to arrive in sequence order. If there are no more responses, or the request’s parameters do not match the next item’s expected request parameters, calls sync_failure_reporter or async_failure_reporter.

For the expected request tuples:

- **method** should be bytes normalized to lowercase.
- **url** should be a str normalized as per the transformations in that (usually) preserve semantics. A URL to http://Something-that-looks-like-a-directory would be normalized to http://Something-that-looks-like-a-directory/ and a URL to http://Something-that-looks-like-a-page/page.html remains unchanged.
- **params** is a dictionary mapping bytes to list of bytes.
- **headers** is a dictionary mapping bytes to list of bytes – note that twisted.web.client.Agent may add its own headers which are not guaranteed to be present (for instance, user-agent or content-length), so it’s better to use some kind of matcher like HasHeaders.
- **data** is a bytes.

For the response tuples:

- **code** is an integer representing the HTTP status code to return.
- **headers** is a dictionary mapping bytes to bytes or str. Note that the value is not a list.
- **body** is a bytes.

Variables

- **sequence** (list) – A sequence of (request tuple, response tuple) two-tuples, as described above.
- **async_failure_reporter** – An optional callable that takes a str message indicating a failure. It’s asynchronous because it cannot just raise an exception—if it does, Resource.render will just convert that into a 500 response, and there will be no other failure reporting mechanism.

When the async_failure_reporter parameter is not passed, async failures will be reported via a twisted.logger.Logger instance, which Trial’s test case classes (twisted.trial.unittest.TestCase and SynchronousTestCase) will translate into a test failure.

**Note:** Some versions of twisted.trial.unittest.SynchronousTestCase report logged errors on the wrong test: see Twisted #9267.

When not subclassing Trial’s classes you must pass async_failure_reporter and implement equivalent behavior or errors will pass silently. For example:
async_failures = []
sequence_stubs = RequestSequence(..., async_failures.append)
stub_treq = StubTreq(StringStubbingResource(sequence_stubs))

with sequence_stubs.consume(self.fail):
    stub_treq.get('http://fakeurl.com')

self.assertEqual([], async_failures)

consume(sync_failure_reporter)

Usage:

sequence_stubs = RequestSequence(...)
stub_treq = StubTreq(StringStubbingResource(sequence_stubs))
# self = twisted.trial.unittest.SynchronousTestCase
with sequence_stubs.consume(self.fail):
    stub_treq.get('http://fakeurl.com')
    stub_treq.get('http://another-fake-url.com')

If there are still remaining expected requests to be made in the sequence, fails the provided test case.

Parameters sync_failure_reporter – A callable that takes a single message reporting failures. This can just raise an exception - it does not need to be asynchronous, since the exception would not get raised within a Resource.

Returns a context manager that can be used to ensure all expected requests have been made.

canceled()  

Returns bool representing whether the entire sequence has been consumed. This is useful in tests to assert that the expected requests have all been made.

StringStubbingResource Objects

class treq.testing.StringStubbingResource(get_response_for)

A resource that takes a callable with 5 parameters (method, url, params, headers, data) and returns (code, headers, body).

The resource uses the callable to return a real response as a result of a request.

The parameters for the callable are:

- method, the HTTP method as bytes.
- url, the full URL of the request as text.
- params, a dictionary of query parameters mapping query keys lists of values (sorted alphabetically).
- headers, a dictionary of headers mapping header keys to a list of header values (sorted alphabetically).
- data, the request body as bytes.

The callable must return a tuple of (code, headers, body) where the code is the HTTP status code, the headers is a dictionary of bytes (unlike the headers parameter, which is a dictionary of lists), and body is a string that will be returned as the response body.

If there is a stubbing error, the return value is undefined (if an exception is raised, Resource will just eat it and return 500 in its place). The callable, or whomever creates the callable, should have a way to handle error reporting.

5.4. The HTTP Client
render(request)
    Produce a response according to the stubs provided.

HasHeaders Objects

class treq.testing.HasHeaders(headers)
    Since Twisted adds headers to a request, such as the host and the content length, it’s necessary to test whether
    request headers CONTAIN the expected headers (the ones that are not automatically added by Twisted).
    This wraps a set of headers, and can be used in an equality test against a superset if the provided headers. The
    headers keys are lowercased, and keys and values are compared in their bytes-encoded forms.
    Headers should be provided as a mapping from strings or bytes to a list of strings or bytes.

5.4.4 MultiPartProducer Objects

treq.multipart.MultiPartProducer is used internally when making requests which involve files.

class treq.multipart.MultiPartProducer(fields,
    boundary=None, 
    cooperator=<module 'twisted.internet.task' from '/home/docs/checkouts/readthedocs.org/user_builds/treq/envs/release-
    22.1.0/lib/python3.8/site-packages/twisted/internet/task.py'>)

MultiPartProducer takes parameters for a HTTP request and produces bytes in multipart/form-data format
defined in RFC 2388 and RFC 2046.

The encoded request is produced incrementally and the bytes are written to a consumer.

Fields should have form: [[(parameter name, value), ...]

Accepted values:

• Unicode strings (in this case parameter will be encoded with utf-8)
• Tuples with (file name, content-type, IBodyProducer objects)

Since MultiPartProducer can accept objects like IBodyProducer which cannot be read from in an event-
    driven manner it uses a Cooperator instance to schedule reads from the underlying producers. Reading is
    also paused and resumed based on notifications from the IConsumer provider being written to.

Variables

• _fields – Sorted parameters, where all strings are enforced to be unicode and file objects
    stacked on bottom (to produce a human readable form-data request)
• _cooperate – A method like Cooperator.cooperate which is used to schedule all reads.
• boundary – The generated boundary used in form-data encoding

pauseProducing()
    Temporarily suspend copying bytes from the input file to the consumer by pausing the CooperativeTask
    which drives that activity.

resumeProducing()
    Undo the effects of a previous pauseProducing and resume copying bytes to the consumer by resuming the
    CooperativeTask which drives the write activity.

startProducing(consumer)
    Start a cooperative task which will read bytes from the input file and write them to consumer. Return a
    Deferred which fires after all bytes have been written.

Parameters consumer – Any IConsumer provider
stopProducing()
Permanently stop writing bytes from the file to the consumer by stopping the underlying CooperativeTask.

5.5 Changelog

5.5.1 22.1.0 (2022-01-29)

Bugfixes

- Cookies specified as a dict were sent to every domain, not just the domain of the request, potentially exposing them on redirect. See GHSA-fhpf-pp6p-55qc. (#339)

5.5.2 21.5.0 (2021-05-24)

Features

- PEP 517/518 build-system metadata is now provided in pyproject.toml. (#329)

Bugfixes

- treq.testing.StubTreq now persists twisted.web.server.Session instances between requests. (#327)

Improved Documentation

- The dependency on Sphinx required to build the documentation has been moved from the dev extra to the new docs extra. (#296)

Deprecations and Removals

- Support for Python 2.7 and 3.5 has been dropped. treq no longer depends on six or mock. (#318)

5.5.3 21.1.0 (2021-01-14)

Features

- Support for Python 3.9: treq is now tested with CPython 3.9. (#305)
- The auth parameter now accepts arbitrary text and bytes for usernames and passwords. Text is encoded as UTF-8, per RFC 7617. Previously only ASCII was allowed. (#268)
- treq produces a more helpful exception when passed a tuple of the wrong size in the files parameter. (#299)
Bugfixes

- The `params` argument once more accepts non-ASCII bytes, fixing a regression first introduced in treq 20.4.1. (#303)
- treq request APIs no longer mutate a `http_headers.Headers` passed as the `headers` parameter when the `auth` parameter is also passed. (#314)
- The agent returned by `treq.auth.add_auth()` and `treq.auth.add_basic_auth()` is now marked to provide `twisted.web.iweb.IAgent`. (#312)
- `treq`’s package metadata has been updated to require `six >= 1.13`, noting a dependency introduced in treq 20.9.0. (#295)

Improved Documentation

- The documentation of the `params` argument has been updated to more accurately describe its type-coercion behavior. (#281)
- The `treq.auth` module has been documented. (#313)

Deprecations and Removals

- Support for Python 2.7, which has reached end of support, is deprecated. This is the last release with support for Python 2.7. (#309)
- Support for Python 3.5, which has reached end of support, is deprecated. This is the last release with support for Python 3.5. (#306)
- Deprecate tolerance of non-string values when passing headers as a dict. They have historically been silently dropped, but will raise TypeError in the next treq release. Also deprecate passing headers other than `dict`, `Headers`, or `None`. Historically falsy values like `[]` or `()` were accepted. (#294)
- `treq` request functions and methods like `treq.get()` and `HTTPClient.post()` now issue a `DeprecationWarning` when passed unknown keyword arguments, rather than ignoring them. Mixing the `json` argument with `files` or `data` is also deprecated. These warnings will change to a `TypeError` in the next treq release. (#297)
- The minimum supported Twisted version has increased to 18.7.0. Older versions are no longer tested in CI. (#307)

5.5.4 20.9.0 (2020-09-27)

Features

- The `url` parameter of `HTTPClient.request()` (and shortcuts like `get()`) now accept `hyperlink.DecodedURL` and `hyperlink.URL` in addition to `str` and `bytes`. (#212)
- Compatibility with the upcoming Twisted 20.9.0 release (#290).
Improved Documentation

- An example of sending and receiving JSON has been added. (#278)

5.5.5 20.4.1 (2020-04-16)

Bugfixes

- Correct a typo in the treq 20.4.0 package metadata that prevented upload to PyPI (pypa/twine#589)

5.5.6 20.4.0 (2020-04-16)

Features

- Support for Python 3.8 and PyPy3: treq is now tested with these interpreters. (#271)

Bugfixes

- `treq.client.HTTPClient.request()` and its aliases no longer raise `UnicodeEncodeError` when passed a Unicode `url` and non-empty `params`. Now the URL and query parameters are concatenated as documented. (#264)
- In treq 20.3.0 the `params` argument didn’t accept parameter names or values that contain the characters `&` or `#`. Now these characters are properly escaped. (#282)

Improved Documentation

- The treq documentation has been revised to emphasize use of `treq.client.HTTPClient` over the module-level convenience functions in the `treq` module. (#276)

5.5.7 20.3.0 (2020-03-15)

Features

- Python 3.7 support. (#228)

Bugfixes

- `treq.testing.RequestTraversalAgent` now passes its memory reactor to the `twisted.web.server.Site` it creates, preventing the `Site` from polluting the global reactor. (#225)
- `treq.testing` no longer generates deprecation warnings about `twisted.test.proto_helpers.MemoryReactor`. (#253)
Improved Documentation

- The `download_file.py` example has been updated to do a streaming download with `unbuffered=True`. (#233)
- The `agent` parameter to `treq.request()` has been documented. (#235)
- The type of the `headers` element of a response tuple passed to `treq.testing.RequestSequence` is now correctly documented as `str`. (#237)

Deprecations and Removals

- Drop support for Python 3.4. (#240)

Misc

- #247, #248, #249
INDICES AND TABLES

• genindex
• modindex
• search
t

treq, 18
trreq.auth, 22
trreq.client, 20
trreq.multipart, 26
trreq.response, 21
trreq.testing, 23
Symbols

_Response (class in treq.response), 21

A
add_auth() (in module treq.auth), 22
add_basic_auth() (in module treq.auth), 22

C
collect() (in module treq), 20
collect() (treq.response._Response method), 21
consume() (treq.testing.RequestSequence method), 25
consumed() (treq.testing.RequestSequence method), 25
content() (in module treq), 20
content() (treq.response._Response method), 21
cookies() (treq.response._Response method), 21

D
delete() (in module treq), 19
delete() (treq.client.HTTPClient method), 21
delete() (treq.testing.treq.testing.StubTreq method), 23
deliverBody() (treq.response._Response method), 22

F
flush() (treq.testing.RequestTraversalAgent method), 23
flush() (treq.testing.treq.testing.StubTreq method), 23

G
get() (in module treq), 19
get() (treq.client.HTTPClient method), 20
get() (treq.testing.treq.testing.StubTreq method), 23

H
HasHeaders (class in treq.testing), 26
head() (in module treq), 19
head() (treq.client.HTTPClient method), 21
head() (treq.testing.treq.testing.StubTreq method), 23
history() (treq.response._Response method), 21
HTTPClient (class in treq.client), 20

J
json() (treq.response._Response method), 21
json_content() (in module treq), 20

M
module
treq, 18
treq.auth, 22
treq.client, 20
treq.multipart, 26
treq.response, 21
treq.testing, 23
MultiPartProducer (class in treq.multipart), 26

P
patch() (in module treq), 19
patch() (treq.client.HTTPClient method), 21
patch() (treq.testing.treq.testing.StubTreq method), 23
pauseProducing() (treq.multipart.MultiPartProducer method), 26
post() (in module treq), 19
post() (treq.client.HTTPClient method), 21
post() (treq.testing.treq.testing.StubTreq method), 23
put() (in module treq), 19
put() (treq.client.HTTPClient method), 21
put() (treq.testing.treq.testing.StubTreq method), 23

R
render() (treq.testing.StringStubbingResource method), 25
request() (in module treq), 18
request() (treq.client.HTTPClient method), 20
request() (treq.testing.RequestTraversalAgent method), 23
request() (treq.testing.treq.testing.StubTreq method), 23
RequestSequence (class in treq.testing), 24
RequestTraversalAgent (class in treq.testing), 23
resumeProducing() (treq.multipart.MultiPartProducer method), 26

RFC
RFC 2046, 26
RFC 2388, 26
RFC 2617, 13
RFC 7231#section-6.4.3, 19
RFC 7617, 22, 27

S

setPreviousResponse() (treq.response._Response method), 22
startProducing() (treq.multipart.MultiPartProducer method), 26
stopProducing() (treq.multipart.MultiPartProducer method), 26
StringStubbingResource (class in treq.testing), 25

T

text() (treq.response._Response method), 21
text_content() (in module treq), 20
treq
    module, 18
treq.auth
    module, 22
treq.client
    module, 20
treq.multipart
    module, 26
treq.response
    module, 21
treq.testing
    module, 23
treq.testing.StubTreq (class in treq.testing), 23

U

UnknownAuthConfig, 22