treq Documentation

Release 18.6.0

David Reid

Jun 04, 2019

Contents

treq depends on a recent Twisted and functions on Python 2.7 and Python 3.3+ (including PyPy).

CHAPTER 1

Why?

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.

CHAPTER 2

Quick Start

Installation:

```
pip install treq
```

2.1 GET

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

Full example: `basic_get.py`

2.2 POST

```
def main(reactor, *args):
    d = treq.post('http://httpbin.org/post',
                  json.dumps({"msg": "Hello!"}).encode('ascii'),
                  headers={b'Content-Type': [b'application/json']})
    d.addCallback(print_response)
    return d
```

Full example: `basic_post.py`

CHAPTER 3

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.

CHAPTER 4

Feature Parity with Requests

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.

	requests	treq
International Domains and URLs	yes	yes
Keep-Alive & Connection Pooling	yes	yes
Sessions with Cookie Persistence	yes	yes
Browser-style SSL Verification	yes	yes
Basic Authentication	yes	yes
Digest Authentication	yes	no
Elegant Key/Value Cookies	yes	yes
Automatic Decompression	yes	yes
Unicode Response Bodies	yes	yes
Multipart File Uploads	yes	yes
Connection Timeouts	yes	yes
.netrc support	yes	no
Python 2.6	yes	no
Python 2.7	yes	yes
Python 3.x	yes	yes

CHAPTER 5

Table of Contents

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.

```
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 Query Parameters

`treq.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 a `str` value or a `list` of `str` values.

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

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

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

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

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

Full example: `query_params.py`

5.1.3 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')`.

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

Full example: `basic_auth.py`

5.1.4 Redirects

treq handles redirects by default.

The following will print a 200 OK response.

```
def main(reactor, *args):
    d = treq.get('http://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.

```
def main(reactor, *args):
    d = treq.get('http://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.

```
def main(reactor, *args):
    d = treq.get('http://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`

5.1.5 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.

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

```
def main(reactor, *args):
    d = treq.get('http://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('http://httpbin.org/cookies', cookies=jar)

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

    return d
```

Full example: `using_cookies.py`

5.1.6 Agent Customization

treq creates its own twisted.web.client.Agent with reasonable defaults, but you may want to provide your own custom agent. A custom agent can be passed to the various treq request methods using the `agent` keyword argument.

```
custom_agent = Agent(reactor, connectTimeout=42)
treq.get(url, agent=custom_agent)
```

Additionally a custom client can be instantiated to use a custom agent using the `agent` keyword argument:

```
custom_agent = Agent(reactor, connectTimeout=42)
client = treq.client.HTTPClient(agent=custom_agent)
client.get(url)
```

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:

```
@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:

```
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

```

(continues on next page)

(continued from previous page)

```
8
9   @defer.inlineCallbacks
10  def make_a_request(treq):
11      """
12      Make a request using treq.
13      """
14      response = yield treq.get('http://an.example/foo', params={'a': 'b'},
15                                headers={b'Accept': b'application/json'})
16      if response.code == http.OK:
17          result = yield response.json()
18      else:
19          message = yield response.text()
20          raise Exception("Got an error from the server: {}".format(message))
21      defer.returnValue(result)
22
23
24  class MakeARequestTests(SynchronousTestCase):
25      """
26      Test :func:`make_a_request()` using :mod:`treq.testing.RequestSequence`.
27      """
28
29      def test_200_ok(self):
30          """On a 200 response, return the response's JSON."""
31          req_seq = RequestSequence([
32              ((b'get', 'http://an.example/foo', {b'a': [b'b']},
33                HasHeaders({'Accept': ['application/json']}), b''),
34               (http.OK, {b'Content-Type': b'application/json'}, b'{"status": "ok"}'))
35          ])
36          treq = StubTreq(StringStubbingResource(req_seq))
37
38          with req_seq.consume(self.fail):
39              result = self.successResultOf(make_a_request(treq))
40
41          self.assertEqual({"status": "ok"}, result)
42
43      def test_418_teapot(self):
44          """On an unexpected response code, raise an exception"""
45          req_seq = RequestSequence([
46              ((b'get', 'http://an.example/foo', {b'a': [b'b']},
47                HasHeaders({'Accept': ['application/json']}), b''),
48               (418, {b'Content-Type': b'text/plain'}, b"I'm a teapot!"))
49          ])
50          treq = StubTreq(StringStubbingResource(req_seq))
51
52          with req_seq.consume(self.fail):
53              failure = self.failureResultOf(make_a_request(treq))
54
55          self.assertEqual(u"Got an error from the server: I'm a teapot!",
56                           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 `mock.ANY`, which compares equal to anything. This sequence matches a single GET request with any parameters or headers:

```
RequestSequence([
    ((b'get', mock.ANY, mock.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

`treq.`**`request`**(*method*, *url*, ***kwargs*)

Make an HTTP request.

Parameters

- **method** (*str*) – HTTP method. Example: `'GET'`, `'HEAD'`. `'PUT'`, `'POST'`.
- **url** (*str*) – 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 as the query string to the URL, any query string parameters in the URL already will be preserved.
- **data** (*str, file-like, IBodyProducer, or None*) – Optional request body.
- **json** (*dict, list/tuple, int, string/unicode, bool, or None*) – Optional JSON-serializable content to pass in body.
- **reactor** – Optional twisted reactor.
- **persistent** (*bool*) – Use persistent HTTP connections. Default: `True`
- **allow_redirects** (*bool*) – Follow HTTP redirects. Default: `True`
- **auth** (tuple of `('username', 'password')`.) – HTTP Basic Authentication information.
- **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 timeframe, a connection is aborted with `CancelledError`.
- **browser_like_redirects** (*bool*) – Use browser like redirects (i.e. Ignore RFC2616 section 10.3 and follow redirects from POST requests). Default: `False`
- **unbuffered** (*bool*) – Pass `True` to to disable response buffering. By default treq buffers the entire response body in memory.
- **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 provider.

`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.

5.3.3 HTTPClient Objects

The *treq.client.HTTPClient* class provides the same interface as the *treq* module itself.

class `treq.client.`**`HTTPClient`**(*agent*, *cookiejar=None*, *data_to_body_producer=<InterfaceClass twisted.web.iweb.IBodyProducer>*)

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

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

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

post(*url*, *data=None*, ***kwargs*)
See *treq.post()*.

put(*url*, *data=None*, ***kwargs*)
See *treq.put()*.

patch(*url*, *data=None*, ***kwargs*)
See *treq.patch()*.

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

5.3.4 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

- **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()

5.3.5 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:

```
[((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):  # self = unittest.TestCase
    stub_treq.get('http://fakeurl.com')

self.assertEqual([], async_failures)
```

consume(***kwds*)

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.

consumed()

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.

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.

5.3.6 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/latest/lib/python2.7/site-packages/twisted/internet/task.pyc'>*)

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 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*.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

S

T