Contents

1 Overview 3
2 Full Documentation 9
3 Indices and tables 39
Python Module Index 41
This is a collection of utilities that some users of python-requests might need but do not belong in requests proper. The library is actively maintained by members of the requests core development team, and so reflects the functionality most requested by users of the requests library.

To get an overview of what the library contains, consult the user documentation.
1.1 requests toolbelt

This is just a collection of utilities for python-requests, but don’t really belong in requests proper. The minimum tested requests version is 2.1.0. In reality, the toolbelt should work with 2.0.1 as well, but some idiosyncracies prevent effective or sane testing on that version.

`pip install requests-toolbelt` to get started!

1.1.1 multipart/form-data Encoder

The main attraction is a streaming multipart form-data object, `MultipartEncoder`. Its API looks like this:

```python
from requests_toolbelt import MultipartEncoder
import requests

m = MultipartEncoder(
    fields={'field0': 'value', 'field1': 'value',
            'field2': ('filename', open('file.py', 'rb'), 'text/plain')}
)

r = requests.post('http://httpbin.org/post', data=m,
                  headers={'Content-Type': m.content_type})
```

You can also use `multipart/form-data` encoding for requests that don’t require files:

```python
from requests_toolbelt import MultipartEncoder
import requests

m = MultipartEncoder(fields={'field0': 'value', 'field1': 'value'})

r = requests.post('http://httpbin.org/post', data=m,
                  headers={'Content-Type': m.content_type})
```
Or, you can just create the string and examine the data:

```python
# Assuming 'm' is one of the above
m.to_string()  # Always returns unicode
```

### 1.1.2 User-Agent constructor

You can easily construct a requests-style User-Agent string:

```python
from requests_toolbelt import user_agent

headers = {
    'User-Agent': user_agent('my_package', '0.0.1')
}
r = requests.get('https://api.github.com/users', headers=headers)
```

### 1.1.3 SSLAdapter

The SSLAdapter was originally published on Cory Benfield’s blog. This adapter allows the user to choose one of the SSL protocols made available in Python’s `ssl` module for outgoing HTTPS connections:

```python
from requests_toolbelt import SSLAdapter
import requests
import ssl

s = requests.Session()
s.mount('https://', SSLAdapter(ssl.PROTOCOL_TLSv1))
```

### 1.1.4 cookies/ForgetfulCookieJar

The ForgetfulCookieJar prevents a particular requests session from storing cookies:

```python
from requests_toolbelt.cookies.forgetful import ForgetfulCookieJar

session = requests.Session()
session.cookies = ForgetfulCookieJar()
```

### 1.1.5 Known Issues

On Python 3.3.0 and 3.3.1, the standard library’s `http` module will fail when passing an instance of the MultipartEncoder. This is fixed in later minor releases of Python 3.3. Please consider upgrading to a later minor version or Python 3.4. *There is absolutely nothing this library can do to work around that bug.*

### 1.1.6 Contributing

Please read the suggested workflow for contributing to this project.
1.2 Contributing to this project

1.2.1 Checklist

1. All potential contributors must read the Contributor Code of Conduct and follow it
2. Fork the repository on GitHub or GitLab
3. Create a new branch, e.g., git checkout -b bug/12345
4. Fix the bug and add tests (if applicable\(^1\), see How To Add Tests)
5. Run the tests (see How To Run The Tests below)
6. Add documentation (as necessary) for your change
7. Build the documentation to check for errors and formatting (see How To Build The Documentation below)
8. Add yourself to the AUTHORS.rst (unless you’re already there)
9. Commit it. Follow these rules in your commit message:
   - Keep the subject line under 50 characters
   - Use an imperative verb to start the commit
   - Use an empty line between the subject and the message
   - Describe the why in detail in the message portion of the commit
   - Wrap the lines of the message at 72 characters
   - Add the appropriate “Closes #12345” syntax to autoclose the issue it fixed (if it closes an issue)
   - See Example Commit Message below
10. Push it to your fork
11. Create a request for us to merge your contribution

After this last step, it is possible that we may leave feedback in the form of review comments. When addressing these comments, you can follow two strategies:
   - Amend/rebase your changes into an existing commit
   - Create a new commit with a different message\(^2\) describing the changes in that commit and push it to your branch

This project is not opinionated about which approach you should prefer. We only ask that you are aware of the following:
   - Neither GitHub nor GitLab notifies us that you have pushed new changes. A friendly ping is encouraged
   - If you continue to use the same branch that you created the request from, both GitHub and GitLab will update the request on the website. You do not need to create a new request for the new changes.

1.2.2 Contributor Code of Conduct

As contributors and maintainers of this project, and in the interest of fostering an open and welcoming community, we pledge to respect all people who contribute through reporting issues, posting feature requests, updating documentation, submitting pull requests or patches, and other activities.

---

\(^1\) You might not need tests if you’re updating documentation, fixing a typo, or updating a docstring. If you’re fixing a bug, please add tests.

\(^2\) If each commit has the same message, the reviewer may ask you to squash your commits or may squash them for you and perform a manual merge.
We are committed to making participation in this project a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, religion, or nationality.

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery
- Personal attacks
- Trolling or insulting/derogatory comments
- Public or private harassment
- Publishing other’s private information, such as physical or electronic addresses, without explicit permission
- Other unethical or unprofessional conduct

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

By adopting this Code of Conduct, project maintainers commit themselves to fairly and consistently applying these principles to every aspect of managing this project. Project maintainers who do not follow or enforce the Code of Conduct may be permanently removed from the project team.

This code of conduct applies both within project spaces and in public spaces when an individual is representing the project or its community.

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting a project maintainer at grafatcolmingov@gmail.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. Maintainers are obligated to maintain confidentiality with regard to the reporter of an incident.

This Code of Conduct is adapted from the Contributor Covenant, version 1.3.0, available at http://contributor-covenant.org/version/1/3/0/

### 1.2.3 How To Add Tests

We use pytest to run tests and to simplify how we write tests. If you’re fixing a bug in an existing please find tests for that module or feature and add to them. Most tests live in the tests directory. If you’re adding a new feature in a new submodule, please create a new module of test code. For example, if you’re adding a submodule named foo then you would create tests/test_foo.py which will contain the tests for the foo submodule.

### 1.2.4 How To Run The Tests

Run the tests in this project using tox. Before you run the tests, ensure you have installed tox either using your system package manager (e.g., apt, yum, etc.), or your prefered python installer (e.g., pip).

Then run the tests on at least Python 2.7 and Python 3.x, e.g.,

```
$ tox -e py27,py34
```

Finally run one, or both, of the flake8 style enforcers, e.g.,

```
$ tox -e py27-flake8
# or
$ tox -e py34-flake8
```
It is preferable if you run both to catch syntax errors that might occur in Python 2 or Python 3 (based on how familiar you are with the common subset of language from both).

Tox will manage virtual environments and dependencies for you so it will be the only dependency you need to install to contribute to this project.

### 1.2.5 How To Build The Documentation

To build the docs, you need to ensure tox is installed and then you may run

```
$ tox -e docs
```

This will build the documentation into `docs/_build/html`. If you then run

```
$ python2.7 -m SimpleHTTPServer
# or
$ python3.4 -m http.server
```

from that directory, you can view the docs locally at `http://localhost:8000/`.

### 1.2.6 Example Commit Message

Allow users to use the frob when uploading data

When uploading data with FooBar, users may need to use the frob method to ensure that pieces of data are not munged.

Closes #1234567

### 1.2.7 Footnotes
2.1 Transport Adapters

The toolbelt comes with several different transport adapters for you to use with requests. The transport adapters are all kept in requests_toolbelt.adapters and include

- requests_toolbelt.adapters.appengine.AppEngineAdapter
- requests_toolbelt.adapters.fingerprint.FingerprintAdapter
- requests_toolbelt.adapters.socket_options.SocketOptionsAdapter
- requests_toolbelt.adapters.socket_options.TCPKeepAliveAdapter
- requests_toolbelt.adapters.source.SourceAddressAdapter
- requests_toolbelt.adapters.ssl.SSLAdapter
- requests_toolbelt.adapters.host_header_ssl.HostHeaderSSLAdapter

2.1.1 AppEngineAdapter

New in version 0.6.0.

As of version 2.10.0, Requests will be capable of supporting Google’s App Engine platform. In order to use Requests on GAE, however, you will need a custom adapter found here as AppEngineAdapter. There are two ways to take advantage of this support at the moment:

1. Using the AppEngineAdapter like every other adapter, e.g.,

```python
import requests
from requests_toolbelt.adapters import appengine

s = requests.Session()
s.mount('http://', appengine.AppEngineAdapter())
s.mount('https://', appengine.AppEngineAdapter())
```
2. By monkey-patching requests to always use the provided adapter:

```python
import requests
from requests_toolbelt.adapters import appengine
appengine.monkeypatch()
```

If you should need to disable certificate validation when monkeypatching (to force third-party libraries that use Requests to not validate certificates, if they do not provide API surface to do so, for example), you can disable it:

```python
from requests_toolbelt.adapters import appengine
appengine.monkeypatch(validate_certificate=False)
```

**Warning:** If `validate_certificate` is `False`, the monkeypatched adapter will *not* validate certificates. This effectively sets the `validate_certificate` argument to `urlfetch.Fetch()` to `False`. You should avoid using this wherever possible. Details can be found in the documentation for `urlfetch.Fetch()`.

```python
class requests_toolbelt.adapters.appengine.AppEngineAdapter(validate_certificate=True, *args, **kwargs)
```

The transport adapter for Requests to use urllib3’s GAE support.

Implements Requests's HTTPAdapter API.

When deploying to Google’s App Engine service, some of Requests’ functionality is broken. There is underlying support for GAE in urllib3. This functionality, however, is opt-in and needs to be enabled explicitly for Requests to be able to use it.

### 2.1.2 FingerprintAdapter

New in version 0.4.0.

By default, requests will validate a server’s certificate to ensure a connection is secure. In addition to this, the user can provide a fingerprint of the certificate they’re expecting to receive. Unfortunately, the requests API does not support this fairly rare use-case. When a user needs this extra validation, they should use the `FingerprintAdapter` class to perform the validation.

```python
class requests_toolbelt.adapters.fingerprint.FingerprintAdapter(fingerprint, **kwargs)
```

A HTTPS Adapter for Python Requests that verifies certificate fingerprints, instead of certificate hostnames.

Example usage:

```python
import requests
import ssl
from requests_toolbelt.adapters.fingerprint import FingerprintAdapter

twitter_fingerprint = '...'
s = requests.Session()
s.mount('https://twitter.com', FingerprintAdapter(twitter_fingerprint))
```

The fingerprint should be provided as a hexadecimal string, optionally containing colons.
2.1.3 SSLAdapter

The SSLAdapter is the canonical implementation of the adapter proposed on Cory Benfield’s blog, here. This adapter allows the user to choose one of the SSL/TLS protocols made available in Python’s ssl module for outgoing HTTPS connections.

In principle, this shouldn’t be necessary: compliant SSL servers should be able to negotiate the required SSL version. In practice there have been bugs in some versions of OpenSSL that mean that this negotiation doesn’t go as planned. It can be useful to be able to simply plug in a Transport Adapter that can paste over the problem.

For example, suppose you’re having difficulty with the server that provides TLS for GitHub. You can work around it by using the following code:

```python
from requests_toolbelt.adapters.ssl import SSLAdapter
import requests
import ssl
s = requests.Session()
s.mount('https://github.com/', SSLAdapter(ssl.PROTOCOL_TLSv1))
```

Any future requests to GitHub made through that adapter will automatically attempt to negotiate TLSv1, and hopefully will succeed.

```python
class requests_toolbelt.adapters.ssl.SSLAdapter(ssl_version=None, **kwargs)
    A HTTPS Adapter for Python Requests that allows the choice of the SSL/TLS version negotiated by Requests. This can be used either to enforce the choice of high-security TLS versions (where supported), or to work around misbehaving servers that fail to correctly negotiate the default TLS version being offered.

Example usage:
```
```python
>>> import requests
>>> import ssl
>>> from requests_toolbelt import SSLAdapter
>>> s = requests.Session()
>>> s.mount('https://', SSLAdapter(ssl.PROTOCOL_TLSv1))
```

You can replace the chosen protocol with any that are available in the default Python SSL module. All subsequent requests that match the adapter prefix will use the chosen SSL version instead of the default.

This adapter will also attempt to change the SSL/TLS version negotiated by Requests when using a proxy. However, this may not always be possible: prior to Requests v2.4.0 the adapter did not have access to the proxy setup code. In earlier versions of Requests, this adapter will not function properly when used with proxies.

2.1.4 HostHeaderSSLAdapter

New in version 0.7.0.

Requests supports SSL Verification by default. However, it relies on the user making a request with the URL that has the hostname in it. If, however, the user needs to make a request with the IP address, they cannot actually verify a certificate against the hostname they want to request.

To accommodate this very rare need, we’ve added HostHeaderSSLAdapter. Example usage:

```python
import requests
from requests_toolbelt.adapters import host_header_ssl
s = requests.Session()
```
class requests_toolbelt.adapters.host_header_ssl.HostHeaderSSLAdapter(pool_connections=10, pool_maxsize=10, max_retries=0, pool_block=False)

A HTTPS Adapter for Python Requests that sets the hostname for certificate verification based on the Host header.
This allows requesting the IP address directly via HTTPS without getting a “hostname doesn’t match” exception.
Example usage:

```python
>>> s.mount('https://', HostHeaderSSLAdapter())
>>> s.get("https://93.184.216.34", headers={"Host": "example.org"})
```

2.1.5 SourceAddressAdapter

New in version 0.3.0.
The `SourceAddressAdapter` allows a user to specify a source address for their connection.

```python
class requests_toolbelt.adapters.source.SourceAddressAdapter(source_address, **kwargs)
```

A Source Address Adapter for Python Requests that enables you to choose the local address to bind to. This allows you to send your HTTP requests from a specific interface and IP address.

Two address formats are accepted. The first is a string: this will set the local IP address to the address given in the string, and will also choose a semi-random high port for the local port number.

The second is a two-tuple of the form (ip address, port): for example, (`10.10.10.10`, 8999). This will set the local IP address to the first element, and the local port to the second element. If 0 is used as the port number, a semi-random high port will be selected.

**Warning:** Setting an explicit local port can have negative interactions with connection-pooling in Requests: in particular, it risks the possibility of getting “Address in use” errors. The string-only argument is generally preferred to the tuple-form.

Example usage:

```python
import requests
from requests_toolbelt.adapters.source import SourceAddressAdapter

s = requests.Session()
s.mount('http://', SourceAddressAdapter('10.10.10.10'))
s.mount('https://', SourceAddressAdapter(('10.10.10.10', 8999)))
```

2.1.6 SocketOptionsAdapter

New in version 0.4.0.
Note: This adapter will only work with requests 2.4.0 or newer. The ability to set arbitrary socket options does not exist prior to requests 2.4.0.

The `SocketOptionsAdapter` allows a user to pass specific options to be set on created sockets when constructing the Adapter without subclassing. The adapter takes advantage of `urllib3`'s support for setting arbitrary socket options for each `urllib3.connection.HTTPConnection` (and `HTTPSConnection`).

To pass socket options, you need to send a list of three-item tuples. For example, `requests` and `urllib3` disable Nagle’s Algorithm by default. If you need to re-enable it, you would do the following:

```python
import socket
import requests
from requests_toolbelt.adapters.socket_options import SocketOptionsAdapter

nagles = [(socket.IPPROTO_TCP, socket.TCP_NODELAY, 0)]
session = requests.Session()
for scheme in session.adapters.keys():
    session.mount(scheme, SocketOptionsAdapter(socket_options=nagles))
```

This would re-enable Nagle’s Algorithm for all `http://` and `https://` connections made with that session.

### Class `requests_toolbelt.adapters.socket_options.SocketOptionsAdapter` (**kwargs)

An adapter for requests that allows users to specify socket options.

Since version 2.4.0 of requests, it is possible to specify a custom list of socket options that need to be set before establishing the connection.

Example usage:

```python
>>> import socket
>>> import requests
>>> from requests_toolbelt.adapters.socket_options import socket_options
>>> s = requests.Session()
>>> opts = [(socket.IPPROTO_TCP, socket.TCP_NODELAY, 0)]
>>> adapter = socket_options.SocketOptionsAdapter(socket_options=opts)
>>> s.mount('http://', adapter)
```

You can also take advantage of the list of default options on this class to keep using the original options in addition to your custom options. In that case, `opts` might look like:

```python
>>> opts = socket_options.SocketOptionsAdapter.default_options + opts
```

### 2.1.7 TCPKeepAliveAdapter

New in version 0.4.0.

Note: This adapter will only work with requests 2.4.0 or newer. The ability to set arbitrary socket options does not exist prior to requests 2.4.0.

The `TCPKeepAliveAdapter` allows a user to pass specific keep-alive related options as keyword parameters as well as arbitrary socket options.

Note: Different keep-alive related socket options may not be available for your platform. Check the socket module for the availability of the following constants:
socket.TCP_KEEPIDLE
socket.TCP_KEEPCNT
socket.TCP_KEEPINTVL

The adapter will silently ignore any option passed for a non-existent option.

An example usage of the adapter:

```python
import requests
from requests_toolbelt.adapters.socket_options import TCPKeepAliveAdapter

session = requests.Session()
keep_alive = TCPKeepAliveAdapter(idle=120, count=20, interval=30)
session.mount('https://region-a.geo-1.compute.hpcloudsvc.com', keep_alive)
session.post('https://region-a.geo-1.compute.hpcloudsvc.com/v2/1234abcdef/servers',
            # ...
)
```

In this case we know that creating a server on HP Public Cloud can cause requests to hang without using TCP Keep-Alive. So we mount the adapter specifically for that domain, instead of adding it to every https:// and http:// request.

**class** requests_toolbelt.adapters.socket_options.TCPKeepAliveAdapter(**kwargs)

An adapter for requests that turns on TCP Keep-Alive by default.

The adapter sets 4 socket options:

- **SOL_SOCKET SO_KEEPALIVE** - This turns on TCP Keep-Alive
- **IPPROTO_TCP TCP_KEEPINTVL 20** - Sets the keep alive interval
- **IPPROTO_TCP TCP_KEEPCNT 5** - Sets the number of keep alive probes
- **IPPROTO_TCP TCP_KEEPIDLE 60** - Sets the keep alive time if the socket library has the TCP_KEEPIDLE constant

The latter three can be overridden by keyword arguments (respectively):

- idle
- interval
- count

You can use this adapter like so:

```python
>>> from requests_toolbelt.adapters import socket_options
>>> tcp = socket_options.TCPKeepAliveAdapter(idle=120, interval=10)
>>> s = requests.Session()
>>> s.mount('http://', tcp)
```

# 2.2 Authentication

requests supports Basic Authentication and HTTP Digest Authentication by default. There are also a number of third-party libraries for authentication with:

- OAuth
- NTLM
requests_toolbelt Documentation, Release 0.8.0

- Kerberos

The requests_toolbelt.auth provides extra authentication features in addition to those. It provides the following authentication classes:

- requests_toolbelt.auth.guess.GuessAuth
- requests_toolbelt.auth.http_proxy_digest.HTTPProxyDigestAuth
- requests_toolbelt.auth.handler.AuthHandler

### 2.2.1 AuthHandler

The AuthHandler is a way of using a single session with multiple websites that require authentication. If you know what websites require a certain kind of authentication and what your credentials are.

Take for example a session that needs to authenticate to GitHub’s API and GitLab’s API, you would set up and use your AuthHandler like so:

```python
import requests
from requests_toolbelt.auth.handler import AuthHandler

def gitlab_auth(request):
    request.headers['PRIVATE-TOKEN'] = 'asecrettoken'

handler = AuthHandler({
    'https://api.github.com': ('sigmavirus24', 'apassword'),
    'https://gitlab.com': gitlab_auth,
})

session = requests.Session()
session.auth = handler
r = session.get('https://api.github.com/user')
# assert r.ok
r2 = session.get('https://gitlab.com/api/v3/projects')
# assert r2.ok
```

**Note:** You must provide both the scheme and domain for authentication. The AuthHandler class will check both the scheme and host to ensure your data is not accidentally exposed.

```python
class requests_toolbelt.auth.handler.AuthHandler стратегий

The AuthHandler object takes a dictionary of domains paired with authentication strategies and will use this to determine which credentials to use when making a request. For example, you could do the following:

```python
from requests import HTTP DigestAuth
from requests_toolbelt.auth.handler import AuthHandler

import requests

auth = AuthHandler({
    'https://api.github.com': ('sigmavirus24', 'fakepassword'),
    'https://example.com': HTTPDigestAuth('username', 'password')
})

r = requests.get('https://api.github.com/user', auth=auth)
# => <Response [200]>
```

(continues on next page)
requests_toolbelt Documentation, Release 0.8.0

```python
r = requests.get('https://example.com/some/path', auth=auth)
# => <Response [200]>

s = requests.Session()
s.auth = auth
r = s.get('https://api.github.com/user')
# => <Response [200]>
```

**Warning:** requests.auth.HTTPDigestAuth is not yet thread-safe. If you use AuthHandler across multiple threads you should instantiate a new AuthHandler for each thread with a new HTTPDigestAuth instance for each thread.

### add_strategy (domain, strategy)
Add a new domain and authentication strategy.

**Parameters**

- **domain** *(str)* – The domain you wish to match against. For example: 'https://api.github.com'

- **strategy** *(str)* – The authentication strategy you wish to use for that domain. For example: ('username', 'password') or requests.HTTPDigestAuth('username', 'password')

```python
a = AuthHandler({})
a.add_strategy('https://api.github.com', ('username', 'password'))
```

### get_strategy_for (url)
Retrieve the authentication strategy for a specified URL.

**Parameters**

- **url** *(str)* – The full URL you will be making a request against. For example, 'https://api.github.com/user'

**Returns** Callable that adds authentication to a request.

```python
import requests
a = AuthHandler({'example.com', ('foo', 'bar')})
strategy = a.get_strategy_for('http://example.com/example')
assert isinstance(strategy, requests.auth.HTTPBasicAuth)
```

### remove_strategy (domain)
Remove the domain and strategy from the collection of strategies.

**Parameters**

- **domain** *(str)* – The domain you wish remove. For example, 'https://api.github.com'.

```python
a = AuthHandler({'example.com', ('foo', 'bar')})
a.remove_strategy('example.com')
assert a.strategies == {}
```

### 2.2.2 GuessAuth

The GuessAuth authentication class automatically detects whether to use basic auth or digest auth:

```python
import requests
a = AuthHandler({'example.com', ('foo', 'bar')})
strategy = a.get_strategy_for('http://example.com/example')
assert isinstance(strategy, requests.auth.HTTPBasicAuth)
```

```python
a = AuthHandler({'example.com', ('foo', 'bar')})
a.remove_strategy('example.com')
assert a.strategies == {}
```
import requests
from requests_toolbelt.auth import GuessAuth

requests.get('http://httpbin.org/basic-auth/user/passwd',
              auth=GuessAuth('user', 'passwd'))
requests.get('http://httpbin.org/digest-auth/auth/user/passwd',
              auth=GuessAuth('user', 'passwd'))

Detection of the auth type is done via the WWW-Authenticate header sent by the server. This requires an additional request in case of basic auth, as usually basic auth is sent preemptively. If the server didn't explicitly require authentication, no credentials are sent.

class requests_toolbelt.auth.guess.GuessAuth(username, password)
    Guesses the auth type by the WWW-Authentication header.

2.2.3 GuessProxyAuth

The GuessProxyAuth handler will automatically detect whether to use basic authentication or digest authentication when authenticating to the provided proxy.

import requests
from requests_toolbelt.auth.guess import GuessProxyAuth

proxies = {
    "http": "http://PROXYSERVER:PROXYPORT",
    "https": "http://PROXYSERVER:PROXYPORT",
}
requests.get('http://httpbin.org/basic-auth/user/passwd',
              auth=GuessProxyAuth('user', 'passwd', 'proxyusr', 'proxypass'),
              proxies=proxies)
requests.get('http://httpbin.org/digest-auth/auth/user/passwd',
              auth=GuessProxyAuth('user', 'passwd', 'proxyusr', 'proxypass'),
              proxies=proxies)

Detection of the auth type is done via the Proxy-Authenticate header sent by the server. This requires an additional request in case of basic auth, as usually basic auth is sent preemptively. If the server didn’t explicitly require authentication, no credentials are sent.

class requests_toolbelt.auth.guess.GuessProxyAuth(username=None, password=None,
                                                    proxy_username=None,
                                                    proxy_password=None)
    Guesses the auth type by WWW-Authentication and Proxy-Authentication headers

2.2.4 HTTPProxyDigestAuth

The HTTPProxyDigestAuth use digest authentication between the client and the proxy.

import requests
from requests_toolbelt.auth.http_proxy_digest import HTTPProxyDigestAuth

proxies = {
    "http": "http://PROXYSERVER:PROXYPORT",
    "https": "https://PROXYSERVER:PROXYPORT",
}
(continues on next page)
url = "https://toolbelt.readthedocs.org/
auth = HTTPProxyDigestAuth("USERNAME", "PASSWORD")
requests.get(url, proxies=proxies, auth=auth)

Program would raise error if the username or password is rejected by the proxy.

```python
class requests_toolbelt.auth.http_proxy_digest.HTTPProxyDigestAuth(*args, **kwargs)
    HTTP digest authentication between proxy

    Parameters stale_rejects (int) – The number of rejects indicate that: the client may wish to
    simply retry the request with a new encrypted response, without reprompting the user for a new
    username and password. i.e., retry build_diggest_header
```

### 2.3 Deprecated Requests Utilities

Requests has decided to deprecate some utility functions in `requests.utils`. To ease users’ lives, they’ve been moved to `requests_toolbelt.utils.deprecated`.

A collection of functions deprecated in `requests.utils`.

```python
import requests
from requests_toolbelt.utils import deprecated
r = requests.get(url)
encodings = deprecated.get_encodings_from_content(r)
```

```python
import requests
from requests_toolbelt.utils import deprecated
r = requests.get(url)
text = deprecated.get_unicode_from_response(r)
```

This will first attempt to retrieve the encoding from the response headers. If that fails, it will use `requests_toolbelt.utils.deprecated.get_encodings_from_content()` to determine encodings from HTML elements.
Parameters `response` *(requests.models.Response)* – Response object to get unicode content from.

### 2.4 Utilities for Downloading Streaming Responses

```python
requests_toolbelt.downloadutils.stream.stream_response_to_file(response, path=None, chunksize=512)
```

Stream a response body to the specified file.

Either use the `path` provided or use the name provided in the `Content-Disposition` header.

**Warning:** If you pass this function an open file-like object as the `path` parameter, the function will not close that file for you.

**Warning:** This function will not automatically close the response object passed in as the `response` parameter.

If a `path` parameter is a directory, this function will parse the `Content-Disposition` header on the response to determine the name of the file as reported by the server, and return a file path in the specified directory. If no `path` parameter is supplied, this function will default to the process’ current working directory.

```python
import requests
from requests_toolbelt import exceptions
from requests_toolbelt.downloadutils import stream

r = requests.get(url, stream=True)
try:
    filename = stream.stream_response_to_file(r)
except exceptions.StreamingError as e:
    # The toolbelt could not find the filename in the
    # Content-Disposition
    print(e.message)
```

You can also specify the filename as a string. This will be passed to the built-in `open()` and we will read the content into the file.

```python
import requests
from requests_toolbelt.downloadutils import stream

r = requests.get(url, stream=True)
filename = stream.stream_response_to_file(r, path='myfile')
```

If the calculated download file path already exists, this function will raise a `StreamingError`.

Instead, if you want to manage the file object yourself, you need to provide either a `io.BytesIO` object or a file opened with the ‘`b`’ flag. See the two examples below for more details.

```python
import requests
from requests_toolbelt.downloadutils import stream

with open('myfile', 'wb') as fd:
```

(continues on next page)
r = requests.get(url, stream=True)
filename = stream.stream_response_to_file(r, path=fd)
print('{0} saved to {1}'.format(url, filename))

import io
import requests
from requests_toolbelt.downloadutils import stream

b = io.BytesIO()
r = requests.get(url, stream=True)
filename = stream.stream_response_to_file(r, path=b)
assert filename is None

Parameters

- **response** (*requests.models.Response*) – A Response object from requests
- **path** (*str*, or object with a `write()` method) – (optional), Either a string with the path to the location to save the response content, or a file-like object expecting bytes.
- **chunksize** (*int*) – (optional), Size of chunk to attempt to stream (default 512B).

Returns

The name of the file, if one can be determined, else None

Return type

*str*

Raises

- `requests_toolbelt.exceptions.StreamingError`

requests_toolbelt.downloadutils.tee.tee

Stream the response both to the generator and a file.

This will stream the response body while writing the bytes to `fileobject`.

Example usage:

```python
resp = requests.get(url, stream=True)
with open('save_file', 'wb') as save_file:
    for chunk in tee(resp, save_file):
        # do stuff with chunk
```

import io

resp = requests.get(url, stream=True)
fileobject = io.BytesIO()

for chunk in tee(resp, fileobject):
    # do stuff with chunk

Parameters

- **response** (*requests.Response*) – Response from requests.
- **chunksize** (*int*) – (optional), Size of chunk to attempt to stream.
- **decode_content** (*bool*) – (optional), If True, this will decode the compressed content of the response.
Raises TypeError if the fileobject wasn’t opened with the right mode or isn’t a BytesIO object.

requests_toolbelt.downloadutils.tee.tee_to_bytearray(response, bytearr, chunksize=65536, decode_content=None)

Stream the response both to the generator and a bytearray.
This will stream the response provided to the function, add them to the provided bytearray and yield them to the user.

Note: This uses the bytearray.extend() by default instead of passing the bytearray into the readinto method.

Example usage:

```python
b = bytearray()
resp = requests.get(url, stream=True)
for chunk in tee_to_bytearray(resp, b):
    # do stuff with chunk
```

Parameters

- `bytearr (bytearray)` – Array to add the streamed bytes to.
- `chunksize (int)` – (optional), Size of chunk to attempt to stream.
- `decode_content (bool)` – (optional), If True, this will decode the compressed content of the response.

requests_toolbelt.downloadutils.tee.tee_to_file(response, filename, chunksize=65536, decode_content=None)

Stream the response both to the generator and a file.
This will open a file named filename and stream the response body while writing the bytes to the opened file object.

Example usage:

```python
resp = requests.get(url, stream=True)
for chunk in tee_to_file(resp, 'save_file'):
    # do stuff with chunk
```

Parameters

- `filename (str)` – Name of file in which we write the response content.
- `chunksize (int)` – (optional), Size of chunk to attempt to stream.
- `decode_content (bool)` – (optional), If True, this will decode the compressed content of the response.
2.5 Utilities for Dumping Information About Responses

Occasionally, it is helpful to know almost exactly what data was sent to a server and what data was received. It can also be challenging at times to gather all of that data from requests because of all of the different places you may need to look to find it. In `requests_toolbelt.utils.dump` there are two functions that will return a `bytearray` with the information retrieved from a response object.

```python
requests_toolbelt.utils.dump.dump_all(response, request_prefix='< ', response_prefix='> ')
```

Dumps all requests and responses including redirects.

This takes the response returned by requests and will dump all request-response pairs in the redirect history in order followed by the final request-response.

Example:

```python
import requests
from requests_toolbelt.utils import dump
dump
resp = requests.get('https://httpbin.org/redirect/5')
data = dump.dump_all(resp)
print(data.decode('utf-8'))
```

**Parameters**

- `response` ([`requests.Response`]) – The response to format
- `request_prefix` ([`bytes`]) – (optional) Bytes to prefix each line of the request data
- `response_prefix` ([`bytes`]) – (optional) Bytes to prefix each line of the response data

**Returns** Formatted bytes of request and response information.

**Return type** `bytearray`

```python
requests_toolbelt.utils.dump.dump_response(response, request_prefix='< ', response_prefix='> ', data_array=None)
```

Dumps a single request-response cycle’s information.

This will take a response object and dump only the data that requests can see for that single request-response cycle.

Example:

```python
import requests
from requests_toolbelt.utils import dump
dump
resp = requests.get('https://api.github.com/users/sigmavirus24')
data = dump.dump_response(resp)
print(data.decode('utf-8'))
```

**Parameters**

- `response` ([`requests.Response`]) – The response to format
- `request_prefix` ([`bytes`]) – (optional) Bytes to prefix each line of the request data
- `response_prefix` ([`bytes`]) – (optional) Bytes to prefix each line of the response data
- `data_array` ([`bytearray`]) – (optional) Bytearray to which we append the request-response cycle data
Returns  Formatted bytes of request and response information.

Return type  bytearray

2.6 Utilities for Enhanced Form-Data Serialization

requests_toolbelt.utils.formdataurlencode(query, *args, **kwargs)
Handle nested form-data queries and serialize them appropriately.

There are times when a website expects a nested form data query to be sent but, the standard library’s urlencode function does not appropriately handle the nested structures. In that case, you need this function which will flatten the structure first and then properly encode it for you.

When using this to send data in the body of a request, make sure you specify the appropriate Content-Type header for the request.

```python
import requests
from requests_toolbelt.utils import formData

query = {
    'my_dict': {
        'foo': 'bar',
        'biz': 'baz',
    },
    'a': 'b',
}

resp = requests.get(url, params=formdataurlencode(query))
# or
resp = requests.post(
    url,
    data=formdata urlencode(query),
    headers={
        'Content-Type': 'application/x-www-form-urlencoded'
    },
)
```

Similarly, you can specify a list of nested tuples, e.g.,

```python
import requests
from requests_toolbelt.utils import formData

query = [
    ('my_list', [
        ('foo', 'bar'),
        ('biz', 'baz'),
    ]),
    ('a', 'b'),
]

resp = requests.get(url, params=formdataurlencode(query))
# or
resp = requests.post(
    url,
    data=formdata urlencode(query),
    headers={
        'Content-Type': 'application/x-www-form-urlencoded'
    },
)
```

(continues on next page)
For additional parameter and return information, see the official urlencode documentation.

### 2.7 Custom Toolbelt Exceptions

Below are the exception classes used by the toolbelt to provide error details to the user of the toolbelt.

Collection of exceptions raised by requests-toolbelt.

- **exception** `requests_toolbelt.exceptions.IgnoringGAECertificateValidation`  
  Used to indicate that given GAE validation behavior will be ignored.
  
  If the user has tried to specify certificate validation when using the insecure AppEngine adapter, it will be ignored (certificate validation will remain off), so we will issue this warning to the user.

- **exception** `requests_toolbelt.exceptions.RequestsVersionTooOld`  
  Used to indicate that the Requests version is too old.
  
  If the version of Requests is too old to support a feature, we will issue this warning to the user.

- **exception** `requests_toolbelt.exceptions.StreamingError`  
  Used in `requests_toolbelt.downloadutils.stream`.

- **exception** `requests_toolbelt.exceptions.VersionMismatchError`  
  Used to indicate a version mismatch in the version of requests required.
  
  The feature in use requires a newer version of Requests to function appropriately but the version installed is not sufficient.

### 2.8 Specialized Sessions

The toolbelt provides specialized session classes in the `requests_toolbelt.sessions` module.

**class** `requests_toolbelt.sessions.BaseUrlSession(base_url=None)`  
A Session with a URL that all requests will use as a base.

Let’s start by looking at an example:

```python
>>> from requests_toolbelt import sessions
>>> s = sessions.BaseUrlSession(  ...
...    base_url='https://example.com/resource/')
>>> r = s.get('/sub-resource/' params={ 'foo': 'bar'})
>>> print(r.request.url)
https://example.com/resource/sub-resource/?foo=bar
```

Our call to the `get` method will make a request to the URL passed in when we created the Session and the partial resource name we provide.

We implement this by overriding the `request` method so most uses of a Session are covered. (This, however, precludes the use of PreparedRequest objects).
Note: The base URL that you provide and the path you provide are very important.

Let's look at another similar example:

```python
>>> from requests_toolbelt import sessions
>>> s = sessions.BaseUrlSession(
    ...    base_url='https://example.com/resource/')
>>> r = s.get('/sub-resource/' params={'foo': 'bar'})
>>> print(r.request.url)
https://example.com/resource/sub-resource/?foo=bar
```

The key difference here is that we called `get` with `/sub-resource/`, i.e., there was a leading `/`. This changes how we create the URL because we rely on `urllib.parse.urljoin`.

To override how we generate the URL, sub-class this method and override the `create_url` method.

Based on implementation from https://github.com/kennethreitz/requests/issues/2554#issuecomment-109341010

```python
create_url(url)
    Create the URL based off this partial path.
```

```python
request(method, url, *args, **kwargs)
    Send the request after generating the complete URL.
```

### 2.8.1 BaseUrlSession

New in version 0.7.0.

Many people have written Session subclasses that allow a “base URL” to be specified so all future requests need not specify the complete URL. To create one simplified and easy to configure version, we’ve added the `requests_toolbelt.sessions.BaseUrlSession` object to the Toolbelt.

```python
class requests_toolbelt.sessions.BaseUrlSession(base_url=None)
    A Session with a URL that all requests will use as a base.
```

Let’s start by looking at an example:

```python
>>> from requests_toolbelt import sessions
>>> s = sessions.BaseUrlSession(
    ...    base_url='https://example.com/resource/')
>>> r = s.get('sub-resource/' params={'foo': 'bar'})
>>> print(r.request.url)
https://example.com/resource/sub-resource/?foo=bar
```

Our call to the `get` method will make a request to the URL passed in when we created the Session and the partial resource name we provide.

We implement this by overriding the `request` method so most uses of a Session are covered. (This, however, precludes the use of PreparedRequest objects).

Note: The base URL that you provide and the path you provide are very important.

Let’s look at another similar example.

2.8. Specialized Sessions
The key difference here is that we called `get` with `/sub-resource/`, i.e., there was a leading `/`. This changes how we create the URL because we rely on `urllib.parse.urljoin`.

To override how we generate the URL, sub-class this method and override the `create_url` method.

```python
create_url(url)
Create the URL based off this partial path.
```

```python
request(method, url, *args, **kwargs)
Send the request after generating the complete URL.
```

## 2.9 Using requests with Threading

New in version 0.4.0.

The toolbelt provides a simple API for using requests with threading.

A requests Session is documented as threadsafe but there are still a couple corner cases where it isn’t perfectly thread-safe. The best way to use a Session is to use one per thread.

The implementation provided by the toolbelt is naïve. This means that we use one session per thread and we make no effort to synchronize attributes (e.g., authentication, cookies, etc.). It also means that we make no attempt to direct a request to a session that has already handled a request to the same domain. In other words, if you’re making requests to multiple domains, the toolbelt’s Pool will not try to send requests to the same domain to the same thread.

This module provides three classes:

- `Pool`
- `ThreadResponse`
- `ThreadException`

In 98% of the situations you’ll want to just use a `Pool` and you’ll treat a `ThreadResponse` as if it were a regular `requests.Response`.

Here’s an example:

```python
# This example assumes Python 3
import queue
from requests_toolbelt.threaded import pool

jobs = queue.Queue()
urls = [
   # My list of URLs to get
]
for url in urls:
   queue.put({'method': 'GET', 'url': url})
(continues on next page)
requests_toolbelt Documentation, Release 0.8.0

(continued from previous page)

```python
p = pool.Pool(job_queue=q)
p.join_all()

for response in p.responses():
    print('GET {0}. Returned {1}.'.format(response.request_kwargs['url'],
                                           response.status_code))
```

This is clearly a bit underwhelming. This is why there’s a short-cut class method to create a `Pool` from a list of URLs.

```python
from requests_toolbelt.threaded import pool

urls = [
    # My list of URLs to get
]
p = pool.Pool.from_urls(urls)
p.join_all()

for response in p.responses():
    print('GET {0}. Returned {1}.'.format(response.request_kwargs['url'],
                                           response.status_code))
```

If one of the URLs in your list throws an exception, it will be accessible from the `exceptions()` generator.

```python
from requests_toolbelt.threaded import pool

urls = [
    # My list of URLs to get
]
p = pool.Pool.from_urls(urls)
p.join_all()

for exc in p.exceptions():
    print('GET {0}. Raised {1}.'.format(exc.request_kwargs['url'],
                                         exc.message))
```

If instead, you want to retry the exceptions that have been raised you can do the following:

```python
from requests_toolbelt.threaded import pool

urls = [
    # My list of URLs to get
]
p = pool.Pool.from_urls(urls)
p.join_all()

new_pool = pool.Pool.from_exceptions(p.exceptions())
new_pool.join_all()
```

Not all requests are advisable to retry without checking if they should be retried. You would normally check if you want to retry it.

The `Pool` object takes 4 other keyword arguments:

- `initializer`
This is a callback that will initialize things on every session created. The callback must return the session.

- **auth_generator**
  This is a callback that is called after the initializer callback has modified the session. This callback must also return the session.

- **num_processes**
  By passing a positive integer that indicates how many threads to use. It is None by default, and will use the result of multiprocessing.cpu_count().

- **session**
  You can pass an alternative constructor or any callable that returns a requests.Session like object. It will not be passed any arguments because a requests.Session does not accept any arguments.

Finally, if you don’t want to worry about Queue or Pool management, you can try the following:

```python
from requests_toolbelt imported threaded

requests = [{
    'method': 'GET',
    'url': 'https://httpbin.org/get',
    # ...
}, {
    # ...
}, {
    # ...
}]

responses_generator, exceptions_generator = threaded.map(requests)

for response in responses_generator:
    # Do something
```

### 2.9.1 API and Module Auto-Generated Documentation

This module provides the API for requests_toolbelt.threaded.

The module provides a clean and simple API for making requests via a thread pool. The thread pool will use sessions for increased performance.

A simple use-case is:

```python
from requests_toolbelt imported threaded

urls_to_get = [{
    'url': 'https://api.github.com/users/sigmavirus24',
    'method': 'GET',
}, {
    'url': 'https://api.github.com/repos/sigmavirus24/requests-toolbelt',
    'method': 'GET',
}, {
    'url': 'https://google.com',
    'method': 'GET',
}]

responses, errors = threaded.map(urls_to_get)
```
By default, the threaded submodule will detect the number of CPUs your computer has and use that if no other number of processes is selected. To change this, always use the keyword argument `num_processes`. Using the above example, we would expand it like so:

```python
responses, errors = threaded.map(urls_to_get, num_processes=10)
```

You can also customize how a `requests.Session` is initialized by creating a callback function:

```python
from requests_toolbelt import user_agent

def initialize_session(session):
    session.headers['User-Agent'] = user_agent('my-scraper', '0.1')
    session.headers['Accept'] = 'application/json'

responses, errors = threaded.map(urls_to_get,
                                 initializer=initialize_session)
```

```python
requests_toolbelt.threaded.map(requests, **kwargs)
```

Simple interface to the threaded Pool object.

This function takes a list of dictionaries representing requests to make using Sessions in threads and returns a tuple where the first item is a generator of successful responses and the second is a generator of exceptions.

**Parameters**

- `requests (list)` – Collection of dictionaries representing requests to make with the Pool object.
- `**kwargs` – Keyword arguments that are passed to the `Pool` object.

**Returns** Tuple of responses and exceptions from the pool

**Return type** `(ThreadResponse, ThreadException)`

Inspiration is blatantly drawn from the standard library’s multiprocessing library. See the following references:

- multiprocessing’s pool source
- map and map_async inspiration

```python
class requests_toolbelt.threaded.pool.Pool(job_queue,
                                          initializer=None,
                                          auth_generator=None, num_processes=None,
                                          session=<class 'requests.sessions.Session'>)
```

Pool that manages the threads containing sessions.

**Parameters**

- `queue (queue.Queue)` – The queue you’re expected to use to which you should add items.
- `initializer (collections.Callable)` – Function used to initialize an instance of `session`.
- `auth_generator (collections.Callable)` – Function used to generate new auth credentials for the session.
- `num_threads (int)` – Number of threads to create.
- `session (requests.Session)`

**exceptions ()**

Iterate over all the exceptions in the pool.

**Returns** Generator of `ThreadException`
classmethod from_exceptions(exceptions, **kwargs)
Create a Pool from an ThreadExceptions.

Provided an iterable that provides ThreadException objects, this classmethod will generate a new pool to retry the requests that caused the exceptions.

Parameters
• exceptions (iterable) – Iterable that returns ThreadException
• kwargs – Keyword arguments passed to the Pool initializer.

Returns An initialized Pool object.
Return type Pool

classmethod from_urls(urls, request_kwargs=None, **kwargs)
Create a Pool from an iterable of URLs.

Parameters
• urls (iterable) – Iterable that returns URLs with which we create a pool.
• request_kwargs (dict) – Dictionary of other keyword arguments to provide to the request method.
• kwargs – Keyword arguments passed to the Pool initializer.

Returns An initialized Pool object.
Return type Pool

get_exception()
Get an exception from the pool.

Return type ThreadException
get_response()
Get a response from the pool.

Return type ThreadResponse
join_all()
Join all the threads to the master thread.

responses()
Iterate over all the responses in the pool.

Returns Generator of ThreadResponse
class requests_toolbelt.threaded.pool.ThreadResponse(request_kwargs, response)
A wrapper around a requests Response object.

This will proxy most attribute access actions to the Response object. For example, if you wanted the parsed JSON from the response, you might do:

```python
thread_response = pool.get_response()
json = thread_response.json()
```

request_kwargs = None
The original keyword arguments provided to the queue

response = None
The wrapped response
class requests_toolbelt.threaded.pool.ThreadException(request_kwargs, exception)

A wrapper around an exception raised during a request.

This will proxy most attribute access actions to the exception object. For example, if you wanted the message
from the exception, you might do:

```python
thread_exc = pool.get_exception()
msg = thread_exc.message
```

exception = None
The captured and wrapped exception

request_kwargs = None
The original keyword arguments provided to the queue

### 2.10 Uploading Data

#### 2.10.1 Streaming Multipart Data Encoder

Requests has support for multipart uploads, but the API means that using that functionality to build exactly the Multi-
part upload you want can be difficult or impossible. Additionally, when using Requests' Multipart upload functionality
all the data must be read into memory before being sent to the server. In extreme cases, this can make it impossible to
send a file as part of a multipart/form-data upload.

The toolbelt contains a class that allows you to build multipart request bodies in exactly the format you need, and to
avoid reading files into memory. An example of how to use it is like this:

```python
import requests
from requests_toolbelt.multipart.encoder import MultipartEncoder

m = MultipartEncoder(fields={'field0': 'value', 'fieldi': 'value',
   'field2': ('filename', open('file.py', 'rb'), 'text/plain'))

r = requests.post('http://httpbin.org/post', data=m,
   headers={'Content-Type': m.content_type})
```

The `MultipartEncoder` has the `.to_string()` convenience method, as well. This method renders the multi-
part body into a string. This is useful when developing your code, allowing you to confirm that the multipart body has
the form you expect before you send it on.

The toolbelt also provides a way to monitor your streaming uploads with the `MultipartEncoderMonitor`.

class requests_toolbelt.multipart.encoder.MultipartEncoder(fields, boundary=None, encoding='utf-8')

The `MultipartEncoder` object is a generic interface to the engine that will create a multipart/form-data body for you.

The basic usage is:

```python
import requests
from requests_toolbelt import MultipartEncoder

encoder = MultipartEncoder({'field': 'value',
```
If you do not need to take advantage of streaming the post body, you can also do:

```python
r = requests.post('https://httpbin.org/post',
    data=encoder.to_string(),
    headers={'Content-Type': encoder.content_type})
```

If you want the encoder to use a specific order, you can use an OrderedDict or more simply, a list of tuples:

```python
coder = MultipartEncoder([('field', 'value'),
                             ('other_field', 'other_value')])
```

2.10.2 Monitoring Your Streaming Multipart Upload

If you need to stream your `multipart/form-data` upload then you’re probably in the situation where it might take a while to upload the content. In these cases, it might make sense to be able to monitor the progress of the upload. For this reason, the toolbelt provides the `MultipartEncoderMonitor`. The monitor wraps an instance of a `MultipartEncoder` and is used exactly like the encoder. It provides a similar API with some additions:

- The monitor accepts a function as a callback. The function is called every time `requests` calls `read` on the monitor and passes in the monitor as an argument.
- The monitor tracks how many bytes have been read in the course of the upload.

You might use the monitor to create a progress bar for the upload. Here is an example using `clint` which displays the progress bar.

To use the monitor you would follow a pattern like this:

```python
import requests
from requests_toolbelt.multipart import encoder

def my_callback(monitor):
    # Your callback function
    pass

e = encoder.MultipartEncoder(
    #...
)
```
If you have a very simple use case you can also do:

```python
import requests
from requests_toolbelt.multipart.encoder import MultipartEncoderMonitor

def my_callback(monitor):
    # Your callback function
    pass

m = MultipartEncoderMonitor.from_fields(
    fields={'field0': 'value', 'field1': 'value',
            'field2': ('filename', open('file.py', 'rb'), 'text/plain')},
    callback=my_callback
)

r = requests.post('http://httpbin.org/post', data=m,
                   headers={'Content-Type': m.content_type})
```

Alternatively, if your use case is very simple, you can use the following pattern.

```python
import requests

def callback(encoder, bytes_read):
    # Do something with this information
    pass

m = MultipartEncoder(fields={'field0': 'value0'})
monitor = MultipartEncoderMonitor(m, callback)
headers = {'Content-Type': monitor.content_type}

r = requests.post('http://httpbin.org/post', data=monitor,
                   headers=headers)
```

2.10. Uploading Data
# Do something with this information
pass

monitor = MultipartEncoderMonitor.from_fields(
    fields={'field0': 'value0'}, callback
)  
headers = {'Content-Type': monitor.content_type}
r = requests.post('https://httpbin.org/post', data=monitor,  
    headers=headers)

2.10.3 Streaming Data from a Generator

There are cases where you, the user, have a generator of some large quantity of data and you already know the size of 
that data. If you pass the generator to requests via the data parameter, requests will assume that you want to 
upload the data in chunks and set a Transfer-Encoding header value of chunked. Often times, this causes the 
server to behave poorly. If you want to avoid this, you can use the StreamingIterator. You pass it the size of 
the data and the generator.

```python
import requests
from requests_toolbelt.streaming_iterator import StreamingIterator
generator = some_function()  # Create your generator
size = some_function_size()  # Get your generator's size
content_type = content_type()  # Get the content-type of the data

streamer = StreamingIterator(size, generator)
r = requests.post('https://httpbin.org/post', data=streamer,  
    headers={'Content-Type': content_type})
```

The streamer will handle your generator for you and buffer the data before passing it to requests.

Changed in version 0.4.0: File-like objects can be passed instead of a generator.

If, for example, you need to upload data being piped into standard in, you might otherwise do:

```python
import requests
import sys

r = requests.post(url, data=sys.stdin)
```

This would stream the data but would use a chunked transfer-encoding. If instead, you know the length of the 
data that is being sent to stdin and you want to prevent the data from being uploaded in chunks, you can use 
the StreamingIterator to stream the contents of the file without relying on chunking.

```python
import requests
from requests_toolbelt.streaming_iterator import StreamingIterator
import sys

stream = StreamingIterator(size, sys.stdin)
r = requests.post(url, data=stream,  
    headers={'Content-Type': content_type})
```

class requests_toolbelt.streaming_iterator.StreamingIterator(size, iterator,  
 encoding='utf-8')

This class provides a way of allowing iterators with a known size to be streamed instead of chunked.
In requests, if you pass in an iterator it assumes you want to use chunked transfer-encoding to upload the data, which not all servers support well. Additionally, you may want to set the content-length yourself to avoid this but that will not work. The only way to preempt requests using a chunked transfer-encoding and forcing it to stream the uploads is to mimic a very specific interface. Instead of having to know these details you can instead just use this class. You simply provide the size and iterator and pass the instance of StreamingIterator to requests via the data parameter like so:

```python
from requests_toolbelt import StreamingIterator
import requests

# Let iterator be some generator that you already have and size be
# the size of the data produced by the iterator
r = requests.post(url, data=StreamingIterator(size, iterator))
```

You can also pass file-like objects to `StreamingIterator` in case requests can’t determine the filesize itself. This is the case with streaming file objects like `stdin` or any sockets. Wrapping e.g. files that are on disk with `StreamingIterator` is unnecessary, because requests can determine the filesize itself.

Naturally, you should also set the `Content-Type` of your upload appropriately because the toolbelt will not attempt to guess that for you.

### 2.11 User-Agent Constructor

Having well-formed user-agent strings is important for the proper functioning of the web. Make server administrators happy by generating yourself a nice user-agent string, just like Requests does! The output of the user-agent generator looks like this:

```python
>>> import requests_toolbelt
>>> requests_toolbelt.user_agent('mypackage', '0.0.1')
'mypackage/0.0.1 CPython/2.7.5 Darwin/13.0.0'
```

The Python type and version, and the platform type and version, will accurately reflect the system that your program is running on. You can drop this easily into your program like this:

```python
from requests_toolbelt import user_agent
from requests import Session

s = Session()
s.headers = {
    'User-Agent': user_agent('my_package', '0.0.1')
}

r = s.get('https://api.github.com/users')
```

This will override the default Requests user-agent string for all of your HTTP requests, replacing it with your own.

#### 2.11.1 Adding Extra Information to Your User-Agent String

New in version 0.5.0.

If you feel it necessary, you can also include versions for other things that your client is using. For example if you were building a package and wanted to include the package name and version number as well as the version of requests and requests-toolbelt you were using you could do the following:
import requests
import requests_toolbelt
from requests_toolbelt.utils import user_agent as ua

user_agent = ua.user_agent('mypackage', '0.0.1',
                           extras=[('requests', requests.__version__),
                                  ('requests-toolbelt', requests_toolbelt.__version__)])

s = requests.Session()
s.headers['User-Agent'] = user_agent

Your user agent will now look like:

mypackage/0.0.1 requests/2.7.0 requests-toolbelt/0.5.0 CPython/2.7.10 Darwin/13.0.0

### 2.11.2 Selecting Only What You Want

New in version 0.8.0.

While most people will find the `user_agent` function sufficient for their usage, others will want to control exactly what information is included in the User-Agent. For those people, the `UserAgentBuilder` is the correct tool. This is the tool that the toolbelt uses inside of `user_agent()`. For example, let’s say you only want your package, its versions, and some extra information, in that case you would do:

```python
import requests
from requests_toolbelt.utils import user_agent as ua

s = requests.Session()
s.headers['User-Agent'] = ua.UserAgentBuilder('mypackage', '0.0.1',
                                               extras=[('requests', requests.__version__),
                                                       ('requests-toolbelt', requests_toolbelt.__version__)]).build()
```

Your user agent will now look like:

mypackage/0.0.1 requests/2.7.0

You can also optionally include the Python version information and System information the same way that our `user_agent` function does.

```python
class requests_toolbelt.utils.user_agent.UserAgentBuilder(name, version)

Class to provide a greater level of control than `user_agent()`.

This is used by `user_agent()` to build its User-Agent string.

user_agent_str = UserAgentBuilder(  
    name='requests-toolbelt',  
    version='17.4.0',  
    ).include_implementation(  
    ).include_system(  
    ).include_extras([  
      ('requests', '2.14.2'),  
      ('urllib3', '1.21.2'),  
    ]).build()
```
build()
Finalize the User-Agent string.

Returns  Formatted User-Agent string.

Return type  str

include_extras(extras)
Include extra portions of the User-Agent.

Parameters extras (list) – list of tuples of extra-name and extra-version

include_implementation()
Append the implementation string to the user-agent string.

This adds the the information that you’re using CPython 2.7.13 to the User-Agent.

include_system()
Append the information about the Operating System.
CHAPTER 3

Indices and tables

- genindex
- modindex
- search
\texttt{r}

\begin{itemize}
  \item \texttt{requests_toolbelt.exceptions}, 24
  \item \texttt{requests_toolbelt.sessions}, 24
  \item \texttt{requests_toolbelt.threaded}, 28
  \item \texttt{requests_toolbelt.utils.deprecated}, 18
\end{itemize}
Index

A
add_strategy() (requests_toolbelt.auth.handler.AuthHandler method), 16
AppEngineAdapter (class in requests_toolbelt.adapters.appengine), 10
AuthHandler (class in requests_toolbelt.auth.handler), 15

B
BaseUrlSession (class in requests_toolbelt.sessions), 24, 25
build() (requests_toolbelt.utils.user_agent.UserAgentBuilder method), 36

C
create_url() (requests_toolbelt.sessions.BaseUrlSession method), 25, 26

D
dump_all() (in module requests_toolbelt.utils.dump), 22
dump_response() (in module requests_toolbelt.utils.dump), 22

E
exception (requests_toolbelt.threaded.pool.ThreadException attribute), 31
exceptions() (requests_toolbelt.threaded.pool.Pool method), 29

F
find_charset() (in module requests_toolbelt.utils.deprecated), 18
find_pragma() (in module requests_toolbelt.utils.deprecated), 18
find_xml() (in module requests_toolbelt.utils.deprecated), 18
FingerprintAdapter (class in requests_toolbelt.adapters.fingerprint), 10
from_exceptions() (requests_toolbelt.threaded.pool.Pool class method), 29
from_urls() (requests_toolbelt.threaded.pool.Pool class method), 30

g
get_encodings_from_content() (in module requests_toolbelt.utils.deprecated), 18
get_exception() (requests_toolbelt.threaded.pool.Pool method), 30
get_response() (requests_toolbelt.threaded.pool.Pool method), 30
get_strategy_for() (requests_toolbelt.auth.handler.AuthHandler method), 16
get_unicode_from_response() (in module requests_toolbelt.utils.deprecated), 18
GuessAuth (class in requests_toolbelt.auth.guess), 17
GuessProxyAuth (class in requests_toolbelt.auth.guess), 17

H
HostHeaderSSLAdapter (class in requests_toolbelt.adapters.host_header_ssl), 12
HTTPProxyDigestAuth (class in requests_toolbelt.auth.http_proxy_digest), 18

I
IgnoringGAECertificateValidation, 24
include_extras() (requests_toolbelt.utils.user_agent.UserAgentBuilder method), 37
include_implementation() (requests_toolbelt.utils.user_agent.UserAgentBuilder method), 37
include_system() (requests_toolbelt.utils.user_agent.UserAgentBuilder method), 37

J
join_all() (requests_toolbelt.threaded.pool.Pool method), 30
map() (in module requests_toolbelt.threaded), 29
MultipartEncoder (class in requests_toolbelt.multipart.encoder), 31
MultipartEncoderMonitor (class in requests_toolbelt.multipart.encoder), 33
Pool (class in requests_toolbelt.threaded.pool), 29
remove_strategy() (requests_toolbelt.auth.handler.AuthHandler method), 16
request() (requests_toolbelt.sessionsBaseUrlSession method), 25, 26
request_kwargs (requests_toolbelt.threaded.pool.ThreadException attribute), 31
request_kwargs (requests_toolbelt.threaded.pool.ThreadResponse attribute), 30
requests_toolbelt.exceptions (module), 24
requests_toolbelt.sessions (module), 24
requests_toolbelt.threaded (module), 28
requests_toolbelt.utils.deprecated (module), 18
RequestsVersionTooOld, 24
response (requests_toolbelt.threaded.pool.ThreadResponse attribute), 30
responses() (requests_toolbelt.threaded.pool.ThreadResponse method), 30
SocketOptionsAdapter (class in requests_toolbelt.adapters.socket_options), 13
SourceAddressAdapter (class in requests_toolbelt.adapters.socket), 12
SSLAdapter (class in requests_toolbelt.adapters.ssl), 11
stream_response_to_file() (in module requests_toolbelt.downloadutils.stream), 19
StreamingError, 24
StreamingIterator (class in requests_toolbelt.streaming_iterator), 34
TCPKeepAliveAdapter (class in requests_toolbelt.adapters.socket_options), 14
tee() (in module requests_toolbelt.downloadutils.tee), 20
tee_to_bytearray() (in module requests_toolbelt.downloadutils.tee), 21
tee_to_file() (in module requests_toolbelt.downloadutils.tee), 21
ThreadException (class in requests_toolbelt.threaded.pool), 30
ThreadResponse (class in requests_toolbelt.threaded.pool), 30
urlencode() (in module requests_toolbelt.utils.formdata), 23
UserAgentBuilder (class in requests_toolbelt.utils.user_agent), 36
VersionMismatchError, 24