transmission-fluid Documentation

Release 0.7-dev

Eric Davis

April 17, 2016

Contents

1 Getting started **3**
1.1 Installing . . . 3
1.2 Creating a client . . . 3

2 First steps **5**
2.1 Dashes in request arguments . . . 5
2.2 Timestamp handling . . . 5

3 Examples **7**
3.1 Getting torrent information . . . 7
3.2 Adding torrents . . . 7
3.3 Removing torrents . . . 7

4 Developers **9**
4.1 Tests and Docs . . . 9
4.2 License . . . 9
4.3 Philosophy . . . 9

5 Changelog **11**
5.1 Version 0.7 (TBD) . . . 11
5.2 Version 0.6 (April 26, 2014) . . . 11
5.3 Version 0.4 (September 7, 2013) . . . 11
5.4 Version 0.3 (August 1, 2012) . . . 11
5.5 Version 0.2.1 (June 24, 2012) . . . 11
5.6 Version 0.2 (June 23, 2012) . . . 11
5.7 Version 0.1 (June 16, 2012) . . . 12

transmission-fluid is a Python wrapper around the Transmission BitTorrent client's RPC interface.

```
>>> from transmission import Transmission
>>> client = Transmission()
>>> response = client('torrent-get', ids=1, fields=['name'])
>>> response['torrents']
[{u'name': u'torrent 1'}]
```

CHAPTER 1

Getting started

transmission-fluid is available on PyPI.

1.1 Installing

Install via *pip* or *easy_install*:

```
$ pip install transmission-fluid
```

Or download the source and run *python setup.py install*, if that's your thing.

transmission-fluid works with Python 2.6, 2.7, and 3.4.

It also depends on the requests library, which will be automatically installed if setuptools or distribute are available.

1.2 Creating a client

Once installed, create a client object:

```
>>> from transmission import Transmission
>>> client = Transmission()
```

By default, the client connects to *localhost:9091* without authentication.

But that can be changed:

```
>>> client = Transmission(host='192.168.1.102', port=9090,
...                       username='foo', password='bar')
```

New in version 0.3: Pass *ssl = True* when constructing the client to use HTTPS for communication between the client and the daemon. It's off by default.

CHAPTER 2

First steps

After creating a client object, communicating with Transmission is a breeze:

```
>>> response = client('torrent-get', ids=[1,2,3], fields=['name'])
>>> response['torrents']
[{u'name': u'torrent 1'}, {u'name': u'torrent 2'}, {u'name': u'torrent 3'}]
```

This calls the torrent-get method along with the *ids* and *fields* request arguments.

No matter what RPC method you're calling, the syntax will always be the same:

client(*method_name*[, ***request_arguments*])

Parameters

- **method_name** (*string*) – A method name as described in RPC specification sections 3.1 to 4.6 (*e.g.,* torrent-start, torrent-add, session-get, etc.)
- **request_arguments** – Keyword arguments as explained in the specification

Return type A dictionary of the "arguments" object returned from Transmission

2.1 Dashes in request arguments

Some request arguments contain dashes in their name. As this is invalid in Python, replace any dashes with underscores:

```
>>> client('torrent-set', ids=1, peer_limit=30) # instead of 'peer-limit'
```

2.2 Timestamp handling

The only area where transmission-fluid deviates from the RPC specification is when dealing with timestamps.

The RPC specification returns all date and time information as Unix timestamps. To make life easier for developers, transmission-fluid transparently converts these timestamps to UTC `datetime` objects:

```
>>> response = client('torrent-get', ids=1, fields=['addedDate'])
>>> response['torrents']
[{u'addedDate': datetime.datetime(2011, 10, 31, 20, 1, 23)}]
```

Similarly, you can pass `datetime` objects as request arguments and they'll be converted to Unix timestamps before being transmitted to Transmission.

New in version 0.3.

CHAPTER 3

Examples

3.1 Getting torrent information

Say you want the name and infohash for the first torrent in Transmission:

```
>>> response = client('torrent-get', ids=1, fields=['name', 'hashString']
>>> response['torrents']
[{u'hashString': u'7c44acebe5828dc53f460c312454141aa3fd1317',
  u'name': 'torrent 1'}]
```

You can also specify a list of IDs:

```
>>> response = client('torrent-get', ids=range(1,11), fields=['name', 'hashString']
>>> response['torrents']
[{u'hashString': u'7c44acebe5828dc53f460c312454141aa3fd1317',
  u'name': 'torrent 1'},
 {u'hashString': u'833e29014ed46e5ea05becc89aaaffb81d0ea9d0',
  u'name': 'torrent 2'}, ... ]
```

ids can also accept infohashes, if you're already working with them:

```
>>> infohash = '833e29014ed46e5ea05becc89aaaffb81d0ea9d0'
>>> response = client('torrent-get', ids=infohash, fields=['name']
>>> response['torrents']
[{u'name': 'torrent 2'}]
```

3.2 Adding torrents

Add a torrent to Transmission by filename:

```
>>> client('torrent-add', filename='/path/to/file.torrent')
```

3.3 Removing torrents

Say you wanted to remove all torrents that were added more than 30 days ago:

```
>>> import datetime
>>> def is_old(torrent):
...   now = datetime.datetime.utcnow()
```

```
...     elapsed = now - torrent['addedDate']
...     return elapsed.days > 30

>>> # When no ids are given, grabs from all torrents
>>> torrents = client('torrent-get', fields=['id', 'addedDate'])['torrents']

>>> torrent_ids = [torrent['id'] for torrent in torrents if is_old(torrent)]
>>> client('torrent-remove', ids=torrent_ids)
```

3.3.1 Operate on torrents in batches

Always try to structure your program to operate on a list of torrent IDs rather than looping through a list of torrents and making an RPC call for each torrent ID.

It's the difference between making one HTTP request with all the torrents you want to operate on and *N* HTTP requests with one ID at a time. The overhead can get noticable quick.

Note: The IDs don't have to be numeric, either. If you're using infohashes already, you can use them as-is in any *ids* request argument.

CHAPTER 4

Developers

transmission-fluid is available on Github and contributions are welcome.

4.1 Tests and Docs

transmission-fluid tries to uphold the Python ecosystem's commitment to well-tested and well-documented code.

As such, use tox to run the full test suite with different Python versions.

Contributions that include tests and docs will likely get acted upon sooner than contributions that don't.

That said, if you have a bugfix or new feature for transmission-fluid but aren't sure how to test it, just explain what you're trying to do. We'll try to work something out. Same goes if English isn't your first language – just do what you can and we'll go from there.

4.2 License

MIT

4.3 Philosophy

The goal of transmission-fluid is to be a thin wrapper around Transmission's RPC specification. It purposefully exposes most of the RPC specification for the following reasons:

Easy for developers Once you've grasped Transmission's RPC specification, you can begin using transmission-fluid immediately. Just plug in a method name and any applicable request arguments and you're ready to go.

Stays current As the Transmission developers add more methods and arguments, you'll be able to use them right away instead of waiting for this wrapper to be updated to take advantage of them.

CHAPTER 5

Changelog

5.1 Version 0.7 (TBD)

5.2 Version 0.6 (April 26, 2014)

- Add Python 3 support

5.3 Version 0.4 (September 7, 2013)

- Bump requests from version 0.13.1 to 1.2.3

5.4 Version 0.3 (August 1, 2012)

- Add SSL proxy support
- Drop *anyjson* requirement
- Automatically convert Unix timestamps to `datetime` objects

5.5 Version 0.2.1 (June 24, 2012)

- Bugfix: Fix *anyjson* ImportError when loading setup.py

5.6 Version 0.2 (June 23, 2012)

- Use setuptools, if available
- Add test suite
- Test against Python 2.6 and 2.7
- Update documentation

5.7 Version 0.1 (June 16, 2012)

- Initial release