curling Documentation

Release 1

Andy McKay

August 21, 2015

Contents

A REST client that wraps slumber to provide a nice interface to tastypie sites for Django.

Curling is under construction right now. It's named curling because of curl and curlish. And I enjoy curling.

Contents:

CHAPTER 1

Installation

From pypi:

```
pip install curling
```

CHAPTER 2

Usage

See the slumber documentation for using slumber. Use curling in the same way:

```
from curling.lib import API
API('http://slumber.in/api/v1/').note.get()
```

Curling sniffs out Tastypie and automatically turns a list of records into a list. For example:

```
res = self.api.services.settings.get()
```

The settings API returns a Tastypie style list. We turn it into a Python list.

We provide a few extra methods for accessing objects that mirror Django methods. For example:

```
res = self.api.services.settings.get_object()
```

Will test that one and only one record is returned and access that object.

class curling.lib.**TastypieResource**(**args*, ***kw*)

get(*data=None*, *headers=None*, ***kwargs*)
Allow a body in GET, because that's just fine.

get_list_or_404(***kw*)
Calls get on a list, returns a 404 if the list isn't there.

Similar to Djangos get_list_or_404.

get_object(***kw*)
Gets an object and checks that one and only one object is returned.

- first it will convert a Tastypie style response into the list of objects.
- **then it will return the first element unless**
  - if there is more than one element raises MultipleObjectsReturned
  - if there is less than one element raises ObjectDoesNotExist
- if a list is not found but another item, that will be returned

get_object_or_404(***kw*)
Wrapper around get_object. The only difference is that if the server returns a HTTP 404, this then alters that into an ObjectDoesNotExist error.

If you've got URLs to items, then *by_url* can be a handy way to access them.

class curling.lib.**CurlingBase**

by_url(*url*, *parser=None*)

Converts a URL such as:

/generic/transaction/ > generic.transaction

And one such as:

/generic/transaction/8/ > generic.transaction(8)

This scheme is assuming that you've got two names and a primary key, if you would like a different parser you could pass in a new one.

2.1 Errors

Just like slumber any response in the 400 range is treated as HttpClientError. We'll also assume that the response body contains JSON and parse that.

So in the example:

```
from lib import API, HttpClientError, HttpServerError

api = API('http://localhost:8001')

try:
    api.by_url('/generic/buyer/').post(data={'foo': 'bar', 'uuid': 'asd'})
except (HttpClientError, HttpServerError), exc:
    print type(exc.content), exc.content
    print type(exc.message), exc.message
```

You'll get:

```
<type 'dict'> {u'uuid': [u'Buyer with this Uuid already exists.']}
<type 'str'> Client Error 400: http://localhost:8001/generic/buyer/
```

- *content*: the body parsed as JSON, if possible
- *message*: the nice message of the error
- *response*: the response object, so *response.status_code* will give you the status

2.2 OAuth

Curling can add in OAuth headers, using OAuth 1.x. These OAuth headers are ones that *solitude <https://solitude.readthedocs.org/en/latest/>* can understand, so might not implement the complete OAuth spec..

For example:

```
api = API('http://localhost:8001', format='jwt')
api.activate_oauth('key', 'secret', realm='optional.realm')
```

You can pass through extra parameters to go through to the oauth generation by using passing through *params*, e.g.:

```
api.activate_oauth('key', 'secret', realm='optional.realm',
                   params={'oauth_token': 'foo'})
```

2.3 Headers

Curling supports optional headers for GET, POST, PUT and PATCH methods. If a GET request contains the *If-None-Match* header with a proper Etag, a 304 response will be returned with an empty content, as expected.

CHAPTER 3

Command

Curling comes with a command line that mirrors curl and curlish, just with a lot less arguments:

```
curling http://slumber.in/api/v1/
{
        "note": {
                "list_endpoint": "/api/v1/note/",
                "schema": "/api/v1/note/schema/"
        }
}
```

3.1 Arguments

All requests are formatted as JSON before being sent to the server and the Content-Type is set to *application/json* so you don't have to set it. Other options:

- -h or –help: show help
- -d or –data: data to be sent, must be valid JSON
- –data-binary: binary data to be sent, cannot be used with –data. When this option is used, Content-Disposition and Content-Type headers are automatically set.
- -X or –request: the verb to be sent, e.g.: curling -X POST to send a POST
- -i or –include: include the HTTP response headers in the output (legacy only)
- -l or –legacy: use the old style command (see below)

The –data and –data-binary options can be the special value '@-', in which case the data is read from stdin.

3.2 Legacy

The legacy command just uses the requests library to make requests, not the actual curling library. Since that's rather daft, I hope to remove that soon.

3.3 Config

This is not available in the legacy code. Curling will use the full curling api from the command line and look for a config file:

- called `.curling` and located in your current directory
- called `.curling` and located in your home directory

It will assume the file is JSON and try and load it. Then it will look up values based on the domain you are trying to access. If `key` and `secret` are present, it will enable oAuth for that URL using those values. The value `realm` is optional. Example config:

```
{
  "marketplace-dev.allizom.org": {
    "key": "mkt:some:key",
    "secret": "yup",
    "realm": "optional.realm"
  }
}
```

3.4 Output

If the response is JSON then the output is pretty printed and syntax highlighted.

If it's HTML and less than 500 characters the output is just displayed in stdout. If it's any other format then the data is saved to a file and automatically opened in a browser (useful for verbose Django error pages).

CHAPTER 4

Developers

4.1 Run The Tests

To hack on Curling, set up a dev environment:

```
pip install -r requirements/tests.txt
pip install -r requirements/docs.txt
```

Run the test suite:

```
nosetests
```

4.2 Build The Docs

```
make -C docs/ html
open docs/build/html/index.html
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

B

C

G

T