webobtoolkit Documentation

Release 0.1.3

Tom Willis

August 16, 2015

Contents

Webobtoolkit is a set of utilities that can be used to compose HTTP clients.

CHAPTER 1

Getting Started with WebObToolKit

Webob toolkit provides an easy way out of the box to interact with web sites or wsgi applications. A webob response is returned for every call so you can leverage your webob knowledge. It may also be useful for people already familiar with WSGI and WSGI middleware.

1.1 The Client

The webobtoolkit client contains a lot of the typical functionality you need in an HTTP client. All current HTTP verbs are supported(GET,POST,PUT,DELETE,OPTIONS,HEAD,...). Each of the methods takes a url, query string, and an optional assert method and returns a webob Response object.

1.1.1 getting a response from a website

Here's an example of how to get a response from wikipedia.org

```
"""
getting a response from wikipedia.org
"""
from webobtoolkit.client import Client
client = Client()
print client.get("http://en.wikipedia.org/wiki/HTTP")
```

1.1.2 getting a response from a WSGI application

Most python web frameworks provide a way to expose your web application as a WSGI app, webobtoolkit can interact with WSGI apps just as if they were running on a web server. This can provide a way for you to unit test your application without the web server overhead.

```
"""
getting a response from a wsgi application
"""
from webobtoolkit import client


def application(environ, start_response):
    """
    most python webframeworks provide a way to expose your web
    application as a WSGI app. consult your framework documentation
    for details.
```

```
    """
    status = "200 OK"  # HTTP messages have a status
    body = "Hello World"  # HTTP messages have a body

    # HTTP messages have headers to describe various things, at a
    # minimum describing the type(Content-Type) and the length of the
    # content(Content-Length)
    headers = [("Content-Type", "text/plain"),
               ("Content-Length",
                str(len(body)))]

    start_response(status, headers)  # calling the function passed in
                                     # with the status and headers of
                                     # the HTTP Response Message

    return [body]  # returning a list containing the body of the HTTP
                   # Response Message

print client.Client(pipeline=client.client_pipeline(application)).get("/")
```

As you can see by the example, all you need to do is construct a client pipeline around your wsgi application. A client pipeline is merely wsgi middleware that handles things that an HTTP client would need to handle like cookies and gzip responses.

1.1.3 parameter passing

Often when interacting with websites or wsgi applications you will need to pass paramters. HTTP provides a couple of ways to do that. One is via query string.

query string

The webobtoolkit client can take a query string as either a string or dictionary like object. Here's an example of using google's ajax search api.

```
"""
passing parameters as a query string
"""
from webobtoolkit.client import Client
client = Client()
result = client.get("http://ajax.googleapis.com/ajax/services/search/web",
                    query_string=dict(v="1.0", q="define: HTTP")).json
for k, v in result.items():
    print k, ":", v
```

form posts

Another way to pass data to a website or wsgi application is through form posts. This example also shows how you might do an assert on the response in order to determine how your logic should proceed.

```
"""
passing parameters as a form post
"""
from webobtoolkit.client import Client
client = Client()
```

```
def assert_success(request, response):
    """
    if response status != 200 then raise an error
    """

    if response.status_int != 200:
        raise Exception("Did not get a valid response from %s" % request.url)


print client.post("http://ajax.googleapis.com/ajax/services/search/web",
                  post=dict(v="1.0", q="define: HTTP"),
                  assert_=assert_success)
```

upload files

WebobToolkit also provides a way to programatically upload files.

```
"""
uploading files example
"""
from webobtoolkit.client import Client, client_pipeline
from webob import Request, Response




def application(environ, start_response):
    """this application merely spits out the keys of the form that was
    posted. we are using webob Request and Response for brevity
    """
    request = Request(environ)
    return Response(str(request.POST.keys()))(environ, start_response)

client = Client(pipeline=client_pipeline(application))
print client.post("/", files=dict(file1=("myfile.txt",
                                         "this is a file containing this text")))
```

1.1.4 built ins

gzip responses

some websites return a response that is compressed in order to reduce bandwidth. By default webobtoolkit can detect and uncompress the responses automatically for you

cookie support

by default webobtoolkit handles cookies and will submit them automatically as part of subsequent requests.

1.1.5 optional logging

The client pipeline has optional logging of both the request and the response. Here's an example of how to enable it.

Once enabled, the request and the response will be logged at whichever log level you specificed.

```
"""
enable logging of request and response
"""
from webobtoolkit import client
import logging
logging.basicConfig(level=logging.DEBUG)

c = client.Client(client.client_pipeline(logging=True, log_level=logging.DEBUG))
c.get("http://google.com")
```

Contents:

CHAPTER 2

Indices and tables

- genindex
- modindex
- search