delver Documentation

Release 0.0.1

Nuncjo

Dec 09, 2017

Contents:

class `delver.`**`Crawler`**(*history=True*, *max_history=5*, *absolute_links=True*)

Browser mimicking object. Mostly wrapper on Requests and Lxml libraries.

Parameters

- **`history`** – (optional) bool, turns off/on history usage in Crawler
- **`max_history`** – (optional) int, max items held in history
- **`absolute_links`** – (optional) bool, makes always all links absolute

Features:

- To some extent, acts like a browser
- Allows visiting pages, form posting, content scraping, cookie handling etc.
- Wraps `requests.Session()`

Simple usage:

```
>>> c = Crawler()
>>> response = c.open('https://httpbin.org/html')
>>> response.status_code
200
```

Form submit:

```
>>> c = Crawler()
>>> response = c.open('https://httpbin.org/forms/post')
>>> forms = c.forms()

Filling up fields values:
>>> form = forms[0]
>>> form.fields = {
...     'custname': 'Ruben Rybnik',
...     'custemail': 'ruben.rybnik@fakemail.com',
...     'size': 'medium',
...     'topping': ['bacon', 'cheese'],
...     'custtel': '+48606505888'
... }
>>> submit_result = c.submit(form)
>>> submit_result.status_code
200

Checking if form post ended with success:
>>> c.submit_check(
...     form,
...     phrase="Ruben Rybnik",
...     url='https://httpbin.org/post',
...     status_codes=[200])
True
```

Form file upload:

```
>>> c = Crawler()
>>> c.open('http://cgi-lib.berkeley.edu/ex/fup.html')
<Response [200]>
>>> forms = c.forms()
>>> upload_form = forms[0]
>>> upload_form.fields = {
...     'note': 'Text file with quote',
```

```
...     'upfile': open('test/test_file.txt', 'r')
... }
>>> c.submit(upload_form, action='http://cgi-lib.berkeley.edu/ex/fup.cgi')
<Response [200]>
>>> c.submit_check(
...     upload_form,
...     phrase="road is easy",
...     status_codes=[200]
... )
True
```

Cookies handling:

```
>>> c = Crawler()
>>> c.open('https://httpbin.org/cookies', cookies={
...     'cookie_1': '1000101000101010',
...     'cookie_2': 'ABABHDBSBAJSLLWO',
... })
<Response [200]>
```

Find links:

```
>>> c = Crawler()
>>> c.open('https://httpbin.org/links/10/0')
<Response [200]>

Links can be filtered by some html tags and filters
like: id, text, title and class:
>>> links = c.links(
...     tags = ('style', 'link', 'script', 'a'),
...     filters = {
...         'text': '7'
...     },
...     match='NOT_EQUAL'
... )
>>> len(links)
8
```

Find images:

```
>>> c = Crawler()
>>> c.open('https://www.python.org/')
<Response [200]>

First image path with 'python-logo' in string:
>>> next(
...     image_path for image_path in c.images()
...     if 'python-logo' in image_path
... )
'https://www.python.org/static/img/python-logo.png'
```

Download file:

```
>>> import os

>>> c = Crawler()
>>> local_file_path = c.download(
...     local_path='test',
```

```
...     url='https://httpbin.org/image/png',
...     name='test.png'
... )
>>> os.path.isfile(local_file_path)
True
```

Download files list in parallel:

```
>>> c = Crawler()
>>> c.open('https://xkcd.com/')
<Response [200]>
>>> full_images_urls = [c.join_url(src) for src in c.images()]
>>> downloaded_files = c.download_files('test', files=full_images_urls)
>>> len(full_images_urls) == len(downloaded_files)
True
```

Traversing through history:

```
>>> c = Crawler()
>>> c.open('http://quotes.toscrape.com/')
<Response [200]>
>>> tags_links = c.links(filters={'class': 'tag'})
>>> c.follow(tags_links[0])
<Response [200]>
>>> c.follow(tags_links[1])
<Response [200]>
>>> c.follow(tags_links[2])
<Response [200]>
>>> history = c.history()
>>> c.back()
>>> c.get_url() == history[-2].url
True
```

add_customized_kwargs(*kwargs*)
Adds request keyword arguments customized by setting *Crawler* attributes like proxy, useragent, headers. Arguments won't be passed if they are already set as *open* method kwargs.

back(*step=1*)
Go back n steps in history, and return response object

clear()
Clears all flow, session, headers etc.

cookies
Wraps *RequestsCookieJar* object from requests library.

Returns *RequestsCookieJar* object

current_parser()
Return parser associated with current flow item.

Returns matched parser object like: class::*HtmlParser* <*HtmlParser*> object

direct_submit(*url=None*, *data=None*)
Direct submit. Used when quick post to form is needed or if there are no forms found by the parser.

Usage:

```
>>> data = {'name': 'Piccolo'}
>>> c = Crawler()
```

```
>>> result = c.submit(action='https://httpbin.org/post', data=data)
>>> result.status_code
200
```

Parameters

- **url** – submit url, form action url, str
- **data** – submit parameters, dict

Returns class::*Response <Response>* object

download_files(*local_path*, *files=None*, *workers=10*)
Download list of files in parallel.

Parameters

- **workers** – number of threads
- **local_path** – download path
- **files** – list of files

Returns list with downloaded files paths

encoding()
Returns current respose encoding.

fit_parser(*response*)
Fits parser according to response type.

Parameters **response** – class::*Response <Response>* object

Returns matched parser object like: class::*HtmlParser <HtmlParser>* object

flow()
Return flow

follow(*url*, *method='get'*, ***kwargs*)
Follows url

forms(*filters=None*)
Return iterable over forms. Doesn't find javascript forms yet (but will be).

example_filters = { 'id': 'searchbox', 'name': 'name, 'action': 'action', 'has_fields': ['field1', 'field2']

}

Usage:

```
>>> c = Crawler()
>>> response = c.open('http://cgi-lib.berkeley.edu/ex/fup.html')
>>> forms = c.forms()
>>> forms[0].fields['note'].get('tag')
'input'
```

forward(*step=1*)
Go forward n steps in history, and return response object

get_url()
Get URL of current document.

handle_response()
Called after request. Make operations accordng to attributes settings.

history()
Return urls history and status codes

join_url(*url_path*)
Returns absolute_url. Path joined with url_root.

open(*url*, *method='get'*, ***kwargs*)
Opens url. Wraps functionality of *Session* from *Requests* library.

Parameters

- **url** – visiting url str
- **method** – 'get', 'post' etc. str
- **kwargs** – additional keywords like headers, cookies etc.

Returns class::*Response <Response>* object

request_history()
Returns current request history (like list of redirects to finally accomplish request)

response()
Get current response.

submit(*form=None*, *action=None*, *data=None*)
Submits form

Parameters

- **form** – *FormWrapper* object
- **action** – custom action url
- **data** – additional custom values to submit

Returns submit result

submit_check(*form*, *phrase=None*, *url=None*, *status_codes=None*)
Checks if success conditions of form submit are met

Parameters

- **form** – *FormWrapper* object
- **phrase** – expected phrase in text
- **url** – expected url
- **status_codes** – list of expected status codes

Returns bool

CHAPTER 1

Main menu

1.1 Installation

This part of the documentation covers the installation of Delver. The first step to using any software package is getting it properly installed.

1.1.1 $ pip install delver

To install Delver, simply run this simple command in your terminal of choice:

```
$ pip install delver
```

1.1.2 Get the Source Code

Delver is actively developed on GitHub, where the code is always available.

You can either clone the public repository:

```
$ git clone git://github.com/nuncjo/Delver.git
```

1.2 Examples

Usage examples can be find in examples.py file or usage examples section file.

Python Module Index

d

Index