# robobrowser Documentation

***Release 0.1***

**Joshua Carp**

February 16, 2014

Contents

Contents:

CHAPTER 1

# RoboBrowser: Your friendly neighborhood web scraper

Homepage: http://robobrowser.readthedocs.org/

RoboBrowser is a simple, Pythonic library for browsing the web without a standalone web browser. RoboBrowser can fetch a page, click on links and buttons, and fill out and submit forms. If you need to interact with web services that don't have APIs, RoboBrowser can help.

```
import re
from robobrowser import RoboBrowser

# Browse to Rap Genius
browser = RoboBrowser(history=True)
browser.open('http://rapgenius.com/')

# Search for Queen
form = browser.get_form(action='/search')
form                # <RoboForm q=>
form['q'].value = 'queen'
browser.submit_form(form)

# Look up the first song
songs = browser.select('.song_name')
browser.follow_link(songs[0])
lyrics = browser.select('.lyrics')
lyrics[0].text      # \n[Intro]\nIs this the real life...

# Back to results page
browser.back()

# Look up my favorite song
browser.follow_link('death on two legs')

# Can also search HTML using regex patterns
lyrics = browser.find(class_=re.compile(r'\blyrics\b'))
lyrics.text         # \n[Verse 1]\nYou suck my blood like a leech...
```

RoboBrowser combines the best of two excellent Python libraries: Requests and BeautifulSoup. RoboBrowser represents browser sessions using Requests and HTML responses using BeautifulSoup, transparently exposing methods of both libraries:

```
import re
from robobrowser import RoboBrowser

browser = RoboBrowser(user_agent='a python robot')
```

```
browser.open('https://github.com/')

# Inspect the browser session
browser.session.cookies['_gh_sess']          # BAh7Bzo...
browser.session.headers['User-Agent']        # a python robot

# Search the parsed HTML
browser.select('div.teaser-icon')        # [<div class="teaser-icon">
                                         # <span class="mega-octicon octicon-checklist"></span>
                                         # </div>,
                                         # ...
browser.find(class_=re.compile(r'column', re.I))     # <div class="one-third column">
                                                     # <div class="teaser-icon">
                                                     # <span class="mega-octicon octicon-checklist"><
                                                     # ...
```

RoboBrowser also includes tools for working with forms, inspired by WebTest and Mechanize.

```
from robobrowser import RoboBrowser

browser = RoboBrowser()
browser.open('http://twitter.com')

# Get the signup form
signup_form = browser.get_form(class_='signup')
signup_form          # <RoboForm user[name]=, user[email]=, ...

# Inspect its values
signup_form['authenticity_token'].value     # 6d03597 ...

# Fill it out
signup_form['user[name]'].value = 'python-robot'
signup_form['user[user_password]'].value = 'secret'

# Serialize it to JSON
signup_form.serialize()          # {'data': {'authenticity_token': '6d03597...',
                                 #  'context': '',
                                 #  'user[email]': '',
                                 #  'user[name]': 'python-robot',
                                 #  'user[user_password]': ''}}

# And submit
browser.submit_form(signup_form)
```

## 1.1 Requirements

- Python >= 2.6 or >= 3.3

## 1.2 License

MIT licensed. See the bundled LICENSE file for more details.

CHAPTER 2

# Installation

At the command line:

```
$ easy_install robobrowser
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv robobrowser
$ pip install robobrowser
```

CHAPTER 3

# History

## 3.1 0.1.0

- First release on PyPI.

CHAPTER 4

# API Reference

## 4.1 browser

Robotic browser

**class** robobrowser.browser.**RoboBrowser**(*auth=None*, *parser=None*, *headers=None*, *user_agent=None*, *history=True*)

Robotic web browser. Represents HTTP requests and responses using the requests library and parsed HTML using BeautifulSoup.

**back**(*n=1*)

Go back in browser history.

**Parameters** **n** (*int*) – Number of pages to go back

**find**

See BeautifulSoup::find.

**find_all**

See BeautifulSoup::find_all.

**follow_link**(*value=None*, **args*, ***kwargs*)

Find a click a link by tag, pattern, and/or BeautifulSoup arguments.

**Parameters** **value** – BeautifulSoup tag, string, or regex. If tag, follow its href; if string or regex, search parsed document for match.

**forward**(*n=1*)

Go forward in browser history.

**Parameters** **n** (*int*) – Number of pages to go forward

**get_form**(*id=None*, **args*, ***kwargs*)

Find form by ID, as well as standard BeautifulSoup arguments.

**Parameters** **id** (*str*) – Form ID

**Returns** BeautifulSoup tag if found, else None

**get_forms**(**args*, ***kwargs*)

Find forms by standard BeautifulSoup arguments.

**Returns** List of BeautifulSoup tags

**get_link**(*text=None*, **args*, ***kwargs*)

Find an anchor or button by containing text, as well as standard BeautifulSoup arguments.

**Parameters** **text** – String or regex to be matched in link text

**Returns** BeautifulSoup tag if found, else None

**get_links**(*text=None*, **args*, ***kwargs*)
Find anchors or buttons by containing text, as well as standard BeautifulSoup arguments.

**Parameters** **text** – String or regex to be matched in link text

**Returns** List of BeautifulSoup tags

**open**(*url*)
Open a URL.

**Parameters** **url** (*str*) – URL

**select**
See `BeautifulSoup::select`.

**submit_form**(*form*)
Submit a form.

**Parameters** **form** (*Form*) – Filled-out form object

**class** robobrowser.browser.**RoboState**(*browser*, *response*)
Representation of a browser state. Wraps the browser and response, and lazily parses the response content.

**parsed**
Lazily parse response content, using HTML parser specified by the browser.

## 4.2 form

HTML forms

**class** robobrowser.forms.form.**Form**(*parsed*)
Representation of an HTML form.

**serialize**()
Serialize each form field and collect the results in a dictionary of dictionaries. Different fields may serialize their contents to different sub-dictionaries: most serialize to data, but file inputs serialize to files. Sub-dictionary keys should correspond to parameters of requests.Request.

**Return dict** Dict-of-dicts of serialized data

## 4.3 fields

HTML form fields

**class** robobrowser.forms.fields.**BaseField**(*parsed*)
Abstract base class for form fields.

**class** robobrowser.forms.fields.**FieldMeta**(*name*, *bases*, *dct*)
Multiply inherit from ValueMeta and ABCMeta; classes with this metaclass are automatically assigned a value property and can use methods fromABCMeta (e.g. abstractmethod).

**mro**() → list
return a type's method resolution order

**register**(*subclass*)
Register a virtual subclass of an ABC.

**class** robobrowser.forms.fields.**ValueMeta**(*name*, *bases*, *dct*)

Metaclass that creates a value property on class creation. Classes with this metaclass should define _get_value and optionally _set_value methods.

**mro**() → list

return a type's method resolution order

CHAPTER 5

# Indices and tables

- *genindex*
- *modindex*
- *search*

## r