.. Documentation

Release 0.4.3

Author

May 31, 2015

Contents

Contents:

CHAPTER 1

Browser

1.1 Basic usages

The browser is a part of oct.core module, and is instantiate by the *GenericTransaction* class in its *__init__* method. The browser can be used as a stand-alone, and for advanced scripts it's good to know how to use it. So how to use it ? First, you need to instantiate a new Browser object:

```
from oct.core.browser import Browser

br = Browser()
```

The *Browser* object takes two optional parameters :

- *sessions* if you want to use custom session manager, default value : *requests.Session()*
- *base_url* for setting up your links when parsing, default to empty string

Now you can use the browser to access urls :

```
response = br.open_url('http://localhost/index.html')
print(response.status_code)
response = br.open_url('http://localhost/other_page.html')
print(response.status_code)
```

This script opens two urls, and for each one display the *status_code* of the response object returned by the *open_url* method.

Since the return value is simply the return of the *requests.get* or of the *requests.post* method, you can access all properties of a basic *requests.Response* object. But we add one thing to it, an *html* property, containing an *lxml.html* object, representing the opened page.

The html property can be used for parsing or getting elements with the *lxml* syntax, since it's a standard object from *lxml.html* parsing.

For example you can access all forms object by using :

```
response.html.forms
```

Or even use the xpath syntax

And can you check the render of the page ? Of course, you don't need other imports, we've implemented an *open_in_browser* static method, calling the *lxml.html.open_in_browser* method. You can use it like this :

```
response = br.open_url('http://localhost/index.html')
br.open_in_browser(response)
```

This will open the page in your default system browser.

A last thing you need to know. Each time the *.html* property is filled, the browser make a call to the *make_links_absolute* method of *lxml*. If you want to avoid that, simply do not provide a *base_url* for your browser instance, it's used only for this call

1.2 Form manipulation

Like we said in the previous part of this documentation, you can use all the *lxml* methods for parsing your page. But again, we have done a part of the job for you.

Let's say that we have a simple html page like this at the index of your localhost favorite web server:

```
<!DOCTYPE html>
<html>

<head>
    <title> My test page </title>
</head>

<body>
    <div id="my_form_block">
        <form action="/action.py" method="post">
            <input type="text" name="firstname" />
        </form>
    </div>
</body>

</html>
```

A very simple page, but it's just for the example.

Now let's say that we want to get this form and submit it from the browser object :

```
from oct.core.browser import Browser

# instantiate the browser
br = Browser(base_url='http://localhost')

# open the url
br.open_url('http://localhost')

# now we getting the form, using css selector
br.get_form(selector='div#my_form_block > form')

# we now have two properties for handling the form
# br.form, containing the lxml for object
# br.form_data, a dict containing all fields and values
# let's just set the value and submit it
br.form_data['firstname'] = 'my name'

# and submit it
response = br.submit_form()

# and check the status code
print(response.status_code)
```

And yes, that's it ! Simple, no ? Thanks to the awesome cssselector python library, getting your forms are now simpler (unless you know nothing about css selectors) but even if we don't want or can not use it, we can still use the *get_form* method, and use the *nr* parameter. The *nr* param simply represent the position of the form in our page. Here, simple we only have one form, so let's update our script :

```
from oct.core.browser import Browser

# instantiate the browser
br = Browser(base_url='http://localhost')

# open the url
br.open_url('http://localhost')

# now we getting the form, using css selector
br.get_form(nr=0)

# we now have two properties for handling the form
# br.form, containing the lxml for object
# br.form_data, a dict containing all fields and values
# let's just set the value and submit it
br.form_data['firstname'] = 'my name'

# and submit it
response = br.submit_form()

# and check the status code
print(response.status_code)
```

And here it is, same result !

For more information about form manipulation, please see the lxml documentation

1.3 More navigation

You can follow links inside the html page like this :

```
from oct.core.browser import Browser

# instantiate the browser
br = Browser(base_url='http://localhost')

# open the url
br.open_url('http://localhost')

# now we can follow any link using css selector or a regex
# the regex will look at the text or the href attribute of the link
response = br.follow_link('a.my_links', '.*this link.*')

# oooops wrong link ! (yeah i know, that's doesn't append in script by try to imagine)
# let's go back
response = br.back() # after this we will be again at the index page
# wait no ! go forward ! it was good
response = br.next() # and here we go
# go back again !
response = br.back()
# access another page now
response = br.open_url('index')
```

```
# going forward ?
br.next() # This will raise an EndOfHistory Exception
```

And that's it ! The *follow_link* method is pretty simple actually, it just finds a link by regex and / or css selector, and then opens the url contained in the *href* attribute of this link.

What about the navigation history ? Well at this point the navigation history is managed by an object, who keep traces of all visited url. The history object tries to fit the beaviour of a standard browser and gave you those methods :

- `back` for going back in the history
- `next` for going the the next element
- `clear_history` for removing all urls of the history

Note: If you use the `back` method, and then open an other url, the `next` method won't be avaible anymore, since

once you open an url, there a no next address yet.

Those methods allow you to navigate more easily with the octbrowser. But if you want to use all the methods / properties of the History object, you can use the `history_object` property of the browser to retreive it. See the history documentation below to see all methods avaibles for the history object

1.4 Module details

1.5 octbrowser.browser module

This file contain the main class for the octbrowser

It represent a simple browser object with all methods

class `octbrowser.browser.`**Browser**(*session=None*, *base_url=''*, ***kwargs*)
Bases: `object`

This class represent a minimal browser. Build on top of lxml awesome library it let you write script for accessing or testing website with python scripts

Parameters

- **session** – The session object to use. If set to None will use requests.Session
- **base_url** – The base url for the website, will append it for every link without a full url
- **history** (*octbrowser.history.base.BaseHistory instance*) – The history object to use. If set to None no history will be stored.

add_header(*name*, *value*)
Allow you to add custom header, one by one. Specify existing name for update Headers will be used by every request

Parameters

- **name** (*str*) – the key of the header
- **value** (*str*) – the associated value

Returns None

back()
Go to the previous url in the history

Returns the Response object

Return type requests.Response

Raises NoPreviousPage, HistoryIsNone

clean_browser()
Clears browser history, session, current page, and form state

self._base_url is unmodified :return: None

clean_session()
This function is called by the core of multi-mechanize. It cleans the session for avoiding cache or cookies errors, or giving false results based on cache

Returns None

clear_history()
Re initialise the history

del_header(*key*)
Try to delete the 'key' of headers property

Parameters **key** (*mixed*) – the key to delete

Returns None

follow_link(*selector*, *url_regex=None*)
Will access the first link found with the selector

Raise: oct.core.exceptions.LinkNotFound

Parameters

- **selector** – a string representing a css selector
- **url_regex** – regex for finding the url, can represent the href attribute or the link content

Returns Response object

forward()
Go to the next url in the history

Returns the Response object

Return type requests.Response

Raises EndOfHistory, HistoryIsNone

get_form(*selector=None*, *nr=0*, *at_base=False*)
Get the form selected by the selector and / or the nr param

Raise:

- oct.core.exceptions.FormNotFoundException
- oct.core.exceptions.NoUrlOpen

Parameters

- **selector** – A css-like selector for finding the form
- **nr** – the index of the form, if selector is set to None, it will search on the hole page
- **at_base** – must be set to true in case of form action is on the base_url page

Returns None

get_html_element(*selector*)
Return a html element as string. The element will be find using the *selector* param

Use this method for get single html elements, if you want to get a list of elements, please use *get_html_elements*

Parameters **selector** (*str*) – a string representing a css selector

Returns a string containing the element, if multiples elements are find, it will concat them

Return type str

get_html_elements(*selector*)
Return a list of lxml.html.HtmlElement matching the *selector* argument

Parameters **selector** (*str*) – a string representing a css selector

Returns a list of lxml.html.HtmlElement of finded elements

Return type list

get_resource(*selector*, *output_dir*, *source_attribute='src'*)
Get a specified ressource and write it to the output dir

Raise: OSError

Parameters

- **selector** (*str*) – a string representing a css selector
- **output_dir** (*str*) – the directory where the ressources will be wright
- **source_attribute** (*str*) – the attribute to retreive the url needed for downloading the ressource

Returns number or resources successfully saved (zero for failure)

get_select_values()
Get the available values of all select and select multiple fields in form

Returns a dict containing all values for each fields

Raises NoFormWaiting

history
Return the actual history list

Returns the history list

Return type list

Raises HistoryIsNone

history_object
Return the actual history object

Returns the _history property

Return type History

static open_in_browser(*response*)
Provide a simple interface for *lxml.html.open_in_browser* function. Be careful, use this function only for debug purpose

Parameters **response** –

Returns

open_url(*url*, *data=None*, ***kwargs*)
Open the given url

Parameters

- **url** – The url to access
- **data** – Data to send. If data is set, the browser will make a POST request

Returns The Response object from requests call

refresh()
Refresh the current page by resending the request

Returns the Response object

Return type requests.Response

Raises NoUrlOpen

set_headers(*headers*)
Setter for headers property

Parameters **headers** (*dict*) – a dict containing all headers

Returns None

submit_form()
Submit the form filled with form_data property dict

Raise: oct.core.exceptions.NoFormWaiting

Returns Response object after the submit

1.6 octbrowser.exceptions module

exception octbrowser.exceptions.**EndOfHistory**
Bases: exceptions.Exception

Raised if the next method of an history is called but the actual page is the last element

exception octbrowser.exceptions.**FormNotFoundException**
Bases: exceptions.Exception

Raised in case of FormNotFound with browser

exception octbrowser.exceptions.**HistoryIsEmpty**
Bases: exceptions.Exception

Raised if the get_current_item method of an history is called but the history is empty

exception octbrowser.exceptions.**HistoryIsNone**
Bases: exceptions.Exception

Raised if the _history property of the browser is set to None and one method using it is called

exception octbrowser.exceptions.**LinkNotFound**
Bases: exceptions.Exception

Raised in case of link not found in current html document

exception octbrowser.exceptions.**NoFormWaiting**
Bases: exceptions.Exception

Raised in case of action required form if no form selected

exception octbrowser.exceptions.**NoPreviousPage**
Bases: exceptions.Exception

Raised if the previous method of an history is called but the actual page is the first element

exception octbrowser.exceptions.**NoUrlOpen**
Bases: exceptions.Exception

Raised in case of no url open but requested inside browser class

exception octbrowser.exceptions.**OctGenericException**
Bases: exceptions.Exception

Provide generic exception for reports

1.7 octbrowser.history module

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

O