oct Documentation

Release 0.3.0

valett_e bene_t philip_l frolic_v leverg_l

November 08, 2015

Contents

Oct stand for Open Charge Tester, the goal of this project is to give you the basics tools for writing simple tests. The tests are simple python scripts that make calls to web page or web service, submit data, login, etc... OCT will give you the tools for easily write your test.

This documentation will provide you basics examples for write your tests, use oct-tools, lunch tests, get the results or even customize the results to fit your needs

Note that the OCT project is in early development and is not suitable for production.

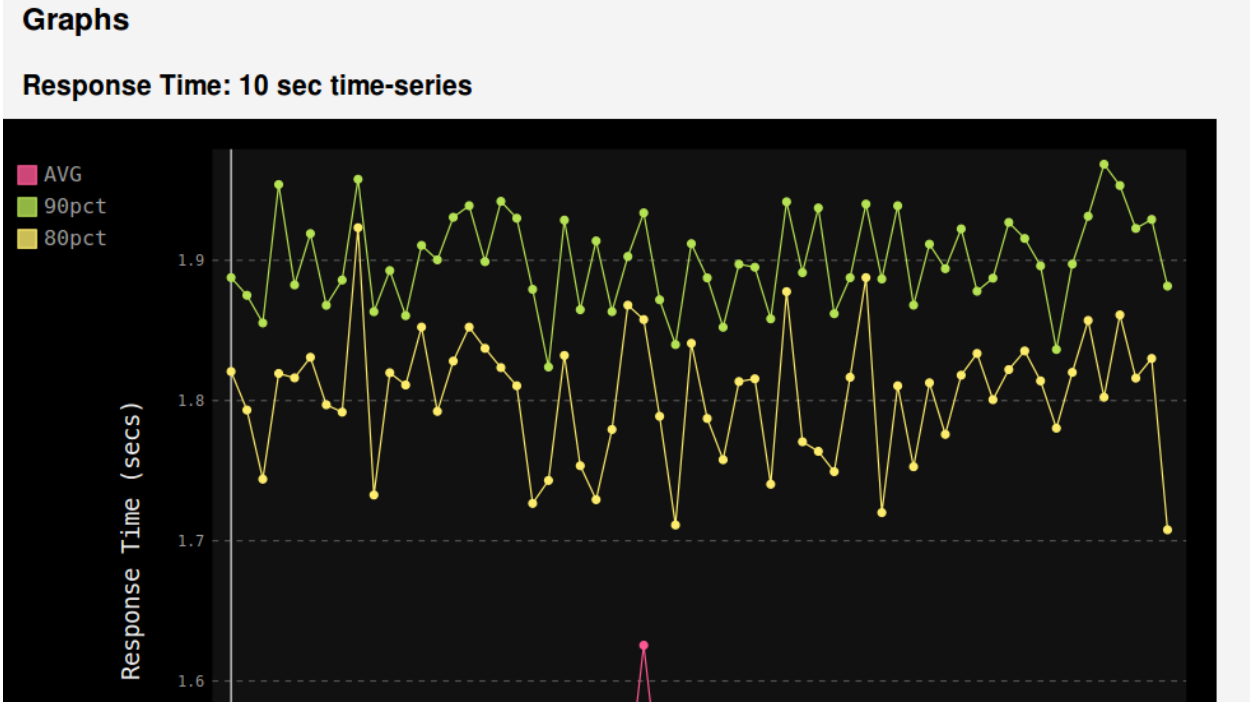
If you want to contribute you're welcome ! Check the git, fork the project, and submit your pull requests !

The OCT module still needs many features at this point, here somme examples :

- Full python3 support (Experimental at the moment)
- Full celery integration for multi-processing
- More generic tests in core module
- More fancy templates
- etc...

CHAPTER 1

Basics module information



OCT is based on multi-mechanize, a library for testing website. But this module is no longer under active development.

So instead of a fork, for building OCT module we include multi-mechanize as a module, and we update it as needed. For the moment modifications are minors and the main job of OCT module is inside the core submodule, which contains a *GenericTransaction* class

> providing you useful methods for writing your tests scripts.

We already have done some update on the multi-mechanize modules like :

- update render of graphics
- update command for new projects
- more information in config file
- customisable templates
- replace matplotlib by pygal for graphics

But other improvements are on the way ! So stay tune on github !

CHAPTER 2

How to

For each functionality, we have tried to write a how to. In that way you should be able to do everything you need with this library, even customize it and add features !

See the `exemples` project page

CHAPTER 3

Installation

You'll need some linux packages for the installation, To install the required packages on Linux systems, use your distribution specific installation tool, e.g. apt-get on Debian/Ubuntu:

```
sudo apt-get install libxml2-dev libxslt-dev python-dev
```

You can install the OCT module with :

```
python setup.py install
```

Or with pip :

```
pip install oct
```

CHAPTER 4

Contents

CHAPTER 5

Indices and tables

5.1 Configuration

This section explain all the configurations available for a project

5.1.1 Basic configuration

The default configuration file look like this

```
[global]
run_time = 30
rampup = 0
results_ts_interval = 10
progress_bar = on
console_logging = off
base_url = http://localhost
default_sleep_time = 2


[user_group-1]
threads = 3
script = v_user.py

[user_group-2]
threads = 3
script = v_user.py
```

5.1.2 Global section

Runtime

```
run_time = 30
```

The time in second for running the project when calling the multimech-run command

Rampup

```
rampup = 0
```

This variable represent the time in second before reaching the specified number of virtual users requested

Results interval

```
results_ts_interval = 10
```

The interval between results in seconds

Progress bar

```
progress_bar = on
```

Set if the progress bar should be shown while running the tests

Logging

```
console_logging = off
```

Set the logging display inside the terminal while running the tests

Base url

```
base_url = http://localhost
```

The base url for the tests. This url will be used by the *open_url* method

Sleep time

```
default_sleep_time = 2
```

The default sleep time in second, used will calling the *user_sleep* method

5.1.3 User group sections

This section defines the virtual groups for testing

Threads

```
threads = 3
```

Define the number of users simulated in this group

5.1.4 Custom configuration variables

In some projects, you may need to have some custom configuration, well that's possible, just add the needed section in the config.cfg file.

Since the *GenericTransaction* class loads the configuration file by default, you can access all the sections and variables you need inside your script.

Let's take a basic configuration file for example :

```
[global]
run_time = 30
rampup = 0
results_ts_interval = 10
progress_bar = on
console_logging = off
base_url = http://localhost
default_sleep_time = 2

[user_group-1]
threads = 3
script = v_user.py

[custom_section]
custom = spam
```

Ok so now inside our test script we want to get the custom value, we just need to do this inside our run method :

```
def run(self):
    spam = self.config.get('custom_section', 'custom')
    print(spam)


if __name__ == '__main__':
    trans = Transaction()
    trans.run()
```

If you run the script, it will display *spam*, since the custom variable value is *spam*

5.2 Writing Scripts

5.2.1 Creating a new project

For starting a new project you have access to this command :

```
oct-newproject <project_name>
```

This command will create a new project inside the current directory named with the <*project_name*> argument

The created directory must look like this :

```
.
-- config.cfg
-- templates
|   -- css
|   |   -- style.css
|   -- footer.html
|   -- head.html
```

```
|   -- img
|   -- scripts
-- test_scripts
    -- v_user.py
```

This folders contains the basic for running an OCT project.

5.2.2 Configuration

For configuration explanation and examples see the Configuration page

5.2.3 Customizing your templates

You need an other render for the results ? the default template is ugly and you want to change it ? It's ok, we have done some things for help you to do that.

If you have created your project with the *oct-newproject* command, you have a templates directory inside your project. This directory is used for writing the results, so each call to *multimech-run* command will read this files. With this you can easily update the template and customize it to fit your needs. It's simple as that.

For the moment the templates can't be fully modified, but you steel have plenty of options to change them.

Let's take a look at the style.css file :

```
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
    display: block;
}
body {
    line-height: 1;
}
```

```
ol, ul {
    list-style: none;
}
blockquote, q {
    quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
    content: '';
    content: none;
}
table {
    border-collapse: collapse;
    border-spacing: 0;
}

body {
    background-color: #f4f4f4;
    font-family: "Helvetica Neue", Helvetica, Roboto, Arial, sans-serif;
}

h1  {
    font-size: 4em;
    background: #2b2b2b;
    color: white;
    font-weight: bold;
}

h2 {
    font-size: 2em;
    background: #f78930;
    margin: 15px 0 15px 0;
}

h1, h2, h3, h4, h5, h6 {
    padding: 15px;
}

h4 {
    font-weight: bold;
    font-size: 1.3em;
}

h3 {
    font-size: 1.5em;
    font-weight: bold;
}

.summary {
    padding-left: 15px;
}

.summary > b {
    font-weight: bold;
}

#main table {
    margin-left: 15px;
```

```
    border: 1px solid grey;
}

#main th {
    font-weight: bold;
    padding: 10px 0 10px 0;
    border: 1px solid grey;
}

#main tr {
    padding: 10px 0 10px 0;
    text-align: center;
}

#main td {
    min-width: 70px;
    padding: 10px 5px 10px 5px;
    border: 1px solid grey;
}

hr {
    color: #f4f4f4;
    background-color: #f4f4f4;
    border: none;
}
```

As you can see, all style present on the result page is here, so feel free to update it. But you may need some other css files, like a css framework, or even javascript files ? why not after all ?

Well you can do that, you can include all the files you need for customize your results page.

How ? simply edit the 'templates/head.html' and include your files, you can even create your own header, add messages at the top of the page, etc...

A little explanation of how this work :

When you call the *multimech-run* command inside your project directory, the command will look for the templates directory and read the *head.html* and the *footer.html* files, and will create a new html page with them. At the same time the command will copy all files insides the *img*, *scripts*, and *css* directories. So everything added in this folders will be in the associated result directory. In that way you can add all the stuff you want to your results, and not reworking each result after each test

5.2.4 Writing your first script

It's time to write our first script and test it, so first let's take a look at the generated v_user.py file :

```
from oct.core.generic import GenericTransaction
import random
import time
import os


CONFIG_PATH = os.path.join(os.path.dirname(os.path.abspath(__file__)), '../')


class Transaction(GenericTransaction):
def __init__(self):
    GenericTransaction.__init__(self, True, CONFIG_PATH)
```

```
def run(self):
    r = random.uniform(1, 2)
    time.sleep(r)
    self.custom_timers['Example_Timer'] = r


if __name__ == '__main__':
trans = Transaction()
trans.run()
print trans.custom_timers
```

So what does this script ? Since it's an example script, actually it just sleep for 1 or 2 seconds.

Let's update this script a little, but first don't forget to update the configuration file to fit your configuration.

Okay so let's write a simple script, just for accessing the index page of our web site and get the statics file of it

```
from oct.core.generic import GenericTransaction
import time
import os


CONFIG_PATH = os.path.join(os.path.dirname(os.path.abspath(__file__)), '../')


class Transaction(GenericTransaction):
    def __init__(self):
        GenericTransaction.__init__(self, True, CONFIG_PATH)

    def run(self):
        test_time = time.time()

        resp = self.open_url('/')

        self.custom_timers['test_time'] = time.time() - test_time


if __name__ == '__main__':
    trans = Transaction()
    trans.run()
    print trans.custom_timers
```

So that's it, we just open the index url of the website (based on the base_url configuration variable)and get the response object returned by the *open_url* method, then set a 'test_time' timer.

So what does this test do ? well it accesses to the index page and retrieve all css, javascript and img files in it. Simple as this

5.2.5 Testing your script

So what's next ? Now you got your basic script retrieving your index page and associated statics files. But does it works ?

Let's figure it out. To test your script 1 time, just to make sure all code work, you actually call the script with your python interpreter like this :

```
python my_script.py
```

With the previous script, if everything is ok, you must see the timer on the standard output.

Everything work find ? Nice, let's now run our tests with lot of users, so update your configuration file and then you just have to run :

```
multimech-run <myproject>
```

Or if you're already inside the path of you're project, simply run :

```
multimech-run .
```

You must see the progress bar appears, you now just have to wait till the tests end. This action will create a results directory inside your project folder, and a sub-directory containing the results in csv or html format.

5.2.6 Handle forms

You now know how to access url and retrieve statics files, but this still basics actions right ? Let's handles some forms and submit some data.

So we gonna take our previous script and update it a bit :

```
from oct.core.generic import GenericTransaction
from oct.testing.content import must_contain
import time
import os


CONFIG_PATH = os.path.join(os.path.dirname(os.path.abspath(__file__)), '../')


class Transaction(GenericTransaction):
    def __init__(self):
        GenericTransaction.__init__(self, True, CONFIG_PATH)

    def run(self):
        test_time = time.time()

        # getting the url
        resp = self.open_url('/showcase/index')

        # now we getting the form, using css selector
        self.get_form(selector='#searchForm')

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

        # getting the response
        resp = self.br.submit_form()

        # checking response content
        must_contain(resp, 'Results that must be found')

        self.custom_timers['test_time'] = time.time() - test_time


if __name__ == '__main__':
```

```
    trans = Transaction()
    trans.run()
    print trans.custom_timers
```

We removed statics management for tests. So what do we do now ? Well let's resume that :

- access the index url
- getting the form with id attribute set to *searchForm*
- considering this form has 1 input named *q*, we set the data for this field to *test*
- submit the form
- checking the content of the returned page.

And that's all, we handle a simple search form, and checking the results !

5.3 oct.core package

5.3.1 oct.core.exceptions module

exception `oct.core.exceptions.`**`FormNotFoundException`**
Bases: `exceptions.Exception`

Raised in case of FormNotFound with browser

exception `oct.core.exceptions.`**`LinkNotFound`**
Bases: `exceptions.Exception`

Raised in case of link not found in current html document

exception `oct.core.exceptions.`**`NoFormWaiting`**
Bases: `exceptions.Exception`

Raised in case of action required form if no form selected

exception `oct.core.exceptions.`**`NoUrlOpen`**
Bases: `exceptions.Exception`

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

exception `oct.core.exceptions.`**`OctConfigurationError`**
Bases: `exceptions.Exception`

Provide an oct configuration error

exception `oct.core.exceptions.`**`OctGenericException`**
Bases: `exceptions.Exception`

Provide generic exception for reports

5.3.2 oct.core.generic module

5.3.3 oct.core.browser module

5.4 oct.multimechanize package

5.4.1 Subpackages

oct.multimechanize.utilities package

Submodules

oct.multimechanize.utilities.gridgui module

Multi-Mechanize Grid Controller sample gui application for controlling multi-mechanize instances via the remote management api

class oct.multimechanize.utilities.gridgui.**Application**(*root*, *hosts*)

check_servers()

clear_window()

get_configs()

get_project_names()

get_results()

list_nodes()

run_tests()

update_configs()

oct.multimechanize.utilities.gridgui.**main**()

oct.multimechanize.utilities.newproject module

oct.multimechanize.utilities.newproject.**create_project**(*project_name*, *config_name='config.cfg'*, *script_name='v_user.py'*, *scripts_dir='test_scripts'*, *config_content='\n[global]\nrun_time = 30\nrampup = 0\nresults_ts_interval = 10\nprogress_bar = on\nconsole_logging = off\nxml_report = off\n\n\n[user_group-1]\nthreads = 3\nscript = v_user.py\n\n[user_group-2]\nthreads = 3\nscript = v_user.py\n\n'*, *script_content="\nimport random\nimport time\n\n\nclass Transaction(object):\n def __init__(self):\n pass\n\n def run(self):\n r = random.uniform(1, 2)\n time.sleep(r)\n self.custom_timers['Example_Timer'] = r\n\n\nif __name__ == '__main__':\n trans = Transaction()\n trans.run()\n print trans.custom_timers\n"*)

oct.multimechanize.utilities.newproject.**main**()

oct.multimechanize.utilities.run module

class oct.multimechanize.utilities.run.**UserGroupConfig**(*num_threads*, *name*, *script_file*)

Bases: object

oct.multimechanize.utilities.run.**main**()

Main function to run multimechanize benchmark/performance test.

oct.multimechanize.utilities.run.**rerun_results**(*project_name*, *cmd_opts*, *results_dir*)

oct.multimechanize.utilities.run.**run**(*project_name*, *cmd_opts*, *remote_starter=None*)

Module contents

5.4.2 Submodules

5.4.3 oct.multimechanize.core module

class oct.multimechanize.core.**Agent**(*queue*, *process_num*, *thread_num*, *start_time*, *run_time*, *user_group_name*, *script_module*, *script_file*)
Bases: threading.Thread

run()

class oct.multimechanize.core.**UserGroup**(*queue*, *process_num*, *user_group_name*, *num_threads*, *script_file*, *run_time*, *rampup*)
Bases: multiprocessing.process.Process

run()

oct.multimechanize.core.**init**(*projects_dir*, *project_name*)
Sanity check that all test scripts can be loaded.

5.4.4 oct.multimechanize.dependency_checker module

script to verify all multi-mechanize dependencies are satisfied

5.4.5 oct.multimechanize.graph module

oct.multimechanize.graph.**resp_graph**(*avg_resptime_points_dict*, *percentile_80_resptime_points_dict*, *percentile_90_resptime_points_dict*, *image_name*, *dir='./'*)

oct.multimechanize.graph.**resp_graph_raw**(*nested_resp_list*, *image_name*, *dir='./'*)

oct.multimechanize.graph.**tp_graph**(*throughputs_dict*, *image_name*, *dir='./'*)

5.4.6 oct.multimechanize.progressbar module

class oct.multimechanize.progressbar.**ProgressBar**(*duration*)
Bases: object

update_time(*elapsed_secs*)

5.4.7 oct.multimechanize.reportwriter module

5.4.8 oct.multimechanize.reportwriterxml module

5.4.9 oct.multimechanize.results module

5.4.10 oct.multimechanize.resultsloader module

5.4.11 oct.multimechanize.resultswriter module

5.4.12 oct.multimechanize.rpcserver module

5.4.13 oct.multimechanize.script_loader module

5.4.14 Module contents

5.5 oct.testing package

5.5.1 Module contents

oct.testing.content.**must_contain**(*resp*, *pattern*)
Test if the pattern is in content

Parameters

- **pattern** (*str*) – pattern to find
- **resp** – a response object

Returns None

Raise AssertionError

oct.testing.content.**must_not_contain**(*resp*, *pattern*)
Test if the pattern is not in content

Parameters

- **pattern** (*str*) – pattern to find
- **resp** – a response object

Returns None

Raise AssertionError

oct.testing.response.**check_response_status**(*resp*, *status*)
This will check is the response_code is equal to the status

Parameters

- **resp** – a response object
- **status** (*int*) – the expected status

Returns None

Raise AssertionError

5.6 oct.tools package

oct.tools contain two functions. One who can be called directly in the shell

```
octtools-user-generator
```

5.6.1 How to

5.6.2 octtools-user-generator

is the command line to generate either user or email WITH their password

occtools-user-generator must have a CSV file provided and have multiple optional arguments

```
MUST HAVE THIS ONE
    -h [CSV File]
[[OPTIONAL]]
    -n [nb_item] Number of items generated
    -s [size] Size of each user/email/password generated
    -w [type u = user, e = email] What you want to generate
```

Default value of each options

```
-n => 250 items
-s => item with lenght of 6
-w => e (generate email by default)
```

5.6.3 Exemple

```
octtools-user-generator userfile.csv -n 25000 -s 6 -w u
```

This command line will generate 25000 email/password with a lenght of 6 in “userfile.csv”

5.6.4 email_generator_func

Is a function with multiple arguments, some have a default value

```
csvfile
what = Define what you want to generate u = user, e = email.
number = Define how many items you want to generate
size = Define the size of each item
chars = Define with 'what' you want to generate you item
```

Default value of each option

```
number = 15
size = 6
char = string.ascii_lowercase
```

5.6.5 Exemple

```
email_generator_func("csvfile.csv", "u", 15000, 7):
```

This command line will generate 15000 user/password with a length of 7 in *csvfile.csv*

5.6.6 oct.tools.email_generator module

oct.tools.email_generator.**email_generator**()
Command line tool for generating csv file containing user / password pairs

Returns None

oct.tools.email_generator.**email_generator_func**(*csvfile*, *what*, *number_of_email=15*, *size=6*, *chars='abcdefghijklmnopqrstuvwxyz'*)

Parameters

- **number_of_email** – Number of random generated email
- **size** – number of char generated
- **chars** – lower the generated char

Type int

Type int

Type string

Returns None

5.6.7 oct.tools.xmltocsv module

oct.tools.xmltocsv.**main**()
Take as options: Xml file, CSV File

Parse the XML and write each value get from it inside the CSV file provided. :return: None

oct.tools.xmltocsv.**sitemap_to_csv**(*xml_file*, *csv_file*)

5.6.8 Module contents

5.7 oct.utilities package

This module provides you basics shell commands to easily use the OCT module.

5.7.1 newproject module

This module provide you a command to create a new project. Once the library is installed you can use :

```
oct-newproject <project-name>
```

It will create a new directory named *project-name*.

The project structure must look like this :

```
.
-- config.cfg
-- templates
|   -- css
|   |   -- style.css
|   -- footer.html
|   -- head.html
|   -- img
|   -- scripts
-- test_scripts
    -- v_user.py
```

With this basic project structure you can start writing your scripts, customize your templates, etc...

This project can be run with the command

```
multimech-run <project>
```

But for test your scripts you can simply run them by using your standard python interpreter

The file *config.cfg* contain all configuration variables for your project. By default it's look like this

```
[global]
run_time = 30
rampup = 0
results_ts_interval = 10
progress_bar = on
console_logging = off
xml_report = off
base_url = http://localhost
default_sleep_time = 2
statics_enabled = 1


[user_group-1]
threads = 3
script = v_user.py

[user_group-2]
threads = 3
script = v_user.py
```

For explanations :

This file give you two virtual user groups, each group has 3 users, and the user script is the *v_user.py* file.

To see all configuration variables explained see the Configuration section

5.7.2 Module doc

oct.utilities.newproject.**create_project**(*project_name*)

oct.utilities.newproject.**main**()

5.7.3 run module

TODO

WORK IN PROGRESS

This module give you access to the command :

```
oct-run <project>
```

This command will run your project using celery. For now the broker can only be configured in source code. It's bind on a standard rabbitmq server.

The goal of this command is to run the same project in several rabbitmq instance.

5.7.4 Module doc

class oct.utilities.run.**UserGroupConfig**(*num_threads*, *name*, *script_file*)
Bases: object

oct.utilities.run.**main**()
Main function to run multimechanize benchmark/performance test.

oct.utilities.run.**rerun_results**(*project_name*, *cmd_opts*, *results_dir*)

oct.utilities.run.**run**(*project_name*, *cmd_opts*, *remote_starter=None*)

5.7.5 celery module

The configuration for running celery

TODO

NOT IMPLEMENTED YET

5.7.6 Module contents

- genindex
- modindex
- search

O

P

R

S

T

U