jsonwspclient Documentation

Release 1.0.3

ellethee

Mar 22, 2018

Contents

jsonwspclient wants to be a simple and, i hope, flexible python client for JSON-WSP services. It is designed for make easy to call services methods and to access to response info and attachments.

JsonWspClient is based on python Requests and uses the Session object. So allows you to persist certain parameters and the cookies across requests.

It supports also *Events handling*, *Response processing*, *Parameters mapping* and *Fault handling*. So you can have a good control over your scripts flow.

Note: **jsonwspclient** is designed to be used with Ladon. and it is based on the original ladon's jsonwsp client.

However it should be flexible enough to be used with other JSON-WSP services.

CHAPTER 1

Contents

1.1 Features

1.1.1 Multiple services

You can load multiple service *descriptions* for a single *JsonWspClient* instance.

So will be more simple make services interact.

```
cli = JsonWspClient(testserver.url, services=['Authenticate', 'TransferService'])
```

1.1.2 Quick service method access

You can call directly the service method as client method (if the name is not already taken).

So you can have a service **Authenticate** with **auth** method and a service **TransferService** with **download** and **upload** methods:

```
cli = JsonWspClient(testserver.url, services=['Authenticate', 'TransferService'])

# which becomes:

cli.auth(...)
cli.donwload(...)
cli.upload(...)
```

Obviously you can access to services and relative methods too (services name will be lowercase).

```
cli = JsonWspClient(testserver.url, services=['Authenticate', 'TransferService'])

cli.authenticate.auth(...)
cli.transferservice.download(...)
cli.transferservice.upload(...)
```

Note: All service methos can accept the `raise_for_fault` parameter which force the response to raise an Exception in case of JSON-WSP fault.

1.1.3 Service methods info

Another thing that could be useful is the method's **info attributes** and **info dictionary**.

info attributes are:

- doc_lines: list of string with doc lines.
- mandatory: list of mandatory parameter names.
- method_name: string with the method name.
- optional: list of optional parameters.
- params_info: dictionary with parameters and relative info.
- params_order: list with parameters order.

And the **info** attribute is a dictionary with all the above information.

```
cli = JsonWspClient(testserver.url, services=['Authenticate'])

print(cli.auth.mandatory)

['username', 'password']
```

1.1.4 Response access:

Every service method call return a *JsonWspResponse* object which is a wrapper for the requests.Reponse object. So you can have all the response things plus some specific features.

The *JsonWspResponse* works in two ways:

- Simple response.
- Multi part response.

When the call to a service method return a simple JSON response **JsonWspResponse** behaves as *simple response* ad you can access only to the `response_dict` and the `result()` attributes which are *interesting*.

When the called method return a *multipart/related* response **JsonWspResponse** behaves as *multi part response* and the methods *next() read_all()* and *save_all()* became usable to access the attachments.

See *Response access:* examples.

1.1.5 Context manager

Both *JsonWspClient* and *JsonWspResponse* supports a basic Context manager protocol. So you can use the **with** Statement.

```
with JsonWspClient('http://mysite.com', services['Authenticate', 'TransferService'])␣
↪as cli:
    with cli.auth(username="name", password="password") as res:
        token = cli.result['token']
```

```
with cli.secure_download(toke=token, name='testfile.txt') as dres:
    if not dres.has_fault:
        dres.save_all('/tmp')
```

1.1.6 Events handling

JsonWspClient handle these events which. Is possible to group events simply by specify only the first part of the event name (it uses the *startwith* to check the event name). Or you can process all events using the * char instead of the event name.

So you can group events using something like `('file.', file_handler)` or `('client.post', mypost)`. Or all events with `('*', all_events)`.

See *Event handling* example.

Note: For all event callbacks only the event_name is mandatory all the other parameters are passed as optional keyword arguments.

---

client

- **client.post.after (event_name, client, path, data, method):**
  - **client:** JsonWspClient instance.
  - **path:** request path relative to the JsonWspClient instance URL.
  - **data:** data passed to the request.
  - **method:** method used for the request.
- **client.post.before (event_name, client, path, data, method):**
  - **client:** JsonWspClient instance.
  - **path:** request path relative to the JsonWspClient instance URL.
  - **data:** data passed to the request.
  - **method:** method used for the request.
- **client.post_mp.after (event_name, client, path, attachs, data, method):**
  - **client:** JsonWspClient instance.
  - **path:** request path relative to the JsonWspClient instance URL.
  - **attachs** Dictionary with attachments.
  - **data:** data passed to the request.
  - **method:** method used for the request.
- **client.post_mp.before (event_name, client, path, attachs, data, method):**
  - **client:** JsonWspClient instance.
  - **path:** request path relative to the JsonWspClient instance URL.
  - **attachs** Dictionary with attachments.
  - **data:** data passed to the request.

  - **method:** method used for the request.

file

- **file.close (event_name, fobj, value, max_value):**
  - **fobj:** file-like object instance.
  - **value:** bytes read/write.
  - **max_value:** file length.
- **file.closed (event_name, fobj, value, max_value):**
  - **fobj:** file-like object instance.
  - **value:** bytes read/write.
  - **max_value:** file length.
- **file.init (event_name, fobj, value, max_value):**
  - **fobj:** file-like object instance.
  - **value:** bytes read/write.
  - **max_value:** file length.
- **file.read (event_name, fobj, value, max_value):**
  - **fobj:** file-like object instance.
  - **value:** bytes read/write.
  - **max_value:** file length.
- **file.write (event_name, fobj, value, max_value):**
  - **fobj:** file-like object instance.
  - **value:** bytes read/write.
  - **max_value:** file length.

service

- **service.call_method.after (event_name, service, method, attachment_map, **kwargs):**
  - **service:** service instance.
  - **method:** called service method name.
  - **attachment_map:** attachment map (if any).
  - ****kwargs:** dictionary with passed params.
- **service.call_method.before (event_name, service, method, attachment_map, **kwargs):**
  - **service:** service instance.
  - **method:** called service method name.
  - **attachment_map:** attachment map (if any).
  - ****kwargs:** dictionary with passed params.
- **service.description_loaded (event_name, service):**

- **service:** service instance.

1.1.7 Response processing

JsonWspClient can process responses before they are returned by the called service method. So you can analyze and/or modify the response on the fly before use it. You can also concatenate multiple **response_processors** obviously all them must return the response object.

```
response_processors(response, service, client, method_name, **kwargs)
```

Note: Only the response is mandatory all the other parameters are passed as optional keyword arguments.

See *Response processing* example.

1.1.8 Parameters mapping

You can also map service methods params to client attributes or methods, string or function. So you can memorize values and silently pass them to services method call as parameters if the method need them.

If you map a parameter with a callable you will receive the method name as first keyword argument and all the other arguments passed to the method (all arguments are optional).

```
def token(method_name, **kwargs):
    """Conditional param"""
    if method_name == 'get_user':
        return '12345'
    return '5678'

cli = JsonWspClient(testserver.url, services=['Authenticate'], params_mapping={'token
↪': token})
```

See *Parameters mapping* example.

1.1.9 Fault handling

JsonWspClient raises automatically response's exceptions (raise_for_status) and *jsonwspexceptions.ParamsError*. However, JSON-WSP errors are normally dealt with silently and are managed by checking the response `has_fault` property. In order for the *JsonWspClient* to raise an exception in case of response fault, you must pass the parameter `raise_for_fault=True` to the client instance or service method. Or use the `raise_for_fault()` method of the response BEFORE using it.

See *Fault handling* example.

Note: In case of parameter `raise_for_fault=True` the response processors are ignored in case of error. While with the `raise_for_fault()` method they are processed BEFORE raising the exception. So, your response processor, must consider it

1.2 Installation

1.3 Examples

Some useful example.

1.3.1 Multiple services and quick method access

```
from jsonwspclient import JsonWspClient

# loads multiple services for mysite.com.
cli = JsonWspClient('http://mysite.com', ['Authenticate', 'TransferService'])
# retrieve the user with the *Authenticate.auth* method.
res = cli.auth(username='username', password='password')
# user is in the response_dict.
user = res.response_dict
# download the file with the *TransferService.download* method
# using the retrieved user-token as credentials.
cli.donwload(token=user['token'], name='testfile.txt').save_all('/tmp')
```

1.3.2 Response access:

Sum numbers and simply print the result.

```
from __future__ import print_function
from jsonwspclient import JsonWspClient

# our client with CalcService.
cli = JsonWspClient('http://mysite.com', ['CalcService'])
# we know our CalcService.sum return a simple int number passing a list of int.
print(cli.sum(numbers=[1, 2, 3]).result)
```

A more complex sum task.

```
from __future__ import print_function
from jsonwspclient import JsonWspClient

# out client with CalcService.
cli = JsonWspClient('http://mysite.com', ['CalcService'])
# we know our CalcService.sum return a simple int number passing a list of int.
# so we will use a list of int list.
numbers_list = [
    [1, 2, 3, 4, 5],
    [10, 20, 5, 7],
    [12, 4, 32, 6],
    [40, 2],
]
# cycle through the number list.
for numbers in numbers_list:
    # print some information.
    print("numbers to add up ", " + ".join([str(a) for a in numbers]))
    # get the sum from our number list from the server.
    with cli.sum(numbers=numbers) as res:
```

```
        # simple test and result print.
        if res.result == 42:
            print("the result is: The answer to the ultimate question of life, the␣
↪universe and everything")
        else:
            print("the result is:", res.result)
```

Cycles over attachments and save.

```
from __future__ import print_function
from jsonwspclient import JsonWspClient

cli = JsonWspClient('http://mysite.com', ['TransferService'])
res = cli.multi_download(names=['test-20-1.txt', 'test-20-2.txt'])
for attach in res:
    filename = "down-file{}".format(attach.index)
    print("Saving", filename)
    attach.save('/tmp/', filename=filename)
```

1.3.3 Event handling

Very simple download monitoring.

```
from __future__ import print_function # python 2
from jsonwspclient import JsonWspClient

# out simple event handler.
def download_event(event_name, fobj, value, max_value):
    """Print event"""
    # print the percentage
    pct = value * float(max_value) / 100
    print("{}%\r".format(pct), end='')

# instantiate out client passing the **download_event** function as handler.
# for the file.read event.
cli = JsonWspClient('http://mysite.com', services=['TransferService'], events=[('file.
↪read', download_event)])
cli.donwload(name='testfile.txt').save_all('/tmp')
```

Deprecation warning on old part in request URL.

```
from jsonwspclient import JsonWspClient

def before_post(event_name, path='', **kwargs):
    """warning"""
    if 'old_request_path' in path:
        raise DeprecationWarning("old_request_path is deprecated, use new_requst_path␣
↪instead")

cli = JsonWspClient('http://mysite.com', services=['TransferService'], events=[(
↪'client.post.before', before_post)])
cli.donwload(name='testfile.txt').save_all('/tmp')
```

See *Events handling*.

1.3.4 Response processing

Imagine we need to authenticate to the server and then keep track of the username and the user token to use them in future service calls. We can achive this easely with the **response processors**.

```
from jsonwspclient import JsonWspClient

# our response_processors
def objectify_result(response, **kwargs):
    """objectify the result"""
    # we add the attribute **result** to the response which will contain the object
    # version of the **result** part of the response_dict.
    response.result = type('Result', (object, ), response.response_dict['result'])
    # we MUST return te response in a processors function.
    return response

def set_user_info(response, service, client, method_name, **kwargs):
    """Set user info if needed"""
    # we check the right service and method:
    if service.name == 'Authenticate' and method_name == 'auth':
        # we concatenated the response_processors se we have the objectifyed result
        # so we can use it and set the client username and token.
        client.username = response.result['username']
        client.token = response.result['token']

# our client with processors.
cli = JsonWspClient('http://mysite.com', services=['Authenticate'],
                    processors=[objectify_result, set_user_info])
# Authenticate
res = cli.auth(username='username', password='password')
# now our client object have the token attribute.
print(cli.token)
```

See *Response processing*.

1.3.5 Parameters mapping

Simple reference to client's attribute mapping.

```
from jsonwspclient import JsonWspClient

# loads multiple services for mysite.com.
cli = JsonWspClient('http://mysite.com', ['Authenticate', 'TransferService'], params_
↪mapping={'token': 'token'})
# retrieve the user with the *Authenticate.auth* method.
res = cli.auth(username='username', password='password')
# set the client attribute token with the result from the request.
cli.token = res.response_dict['result']['token']
# download the file with the *TransferService.download* method
# notice we don't need to pass the token argument because now is mapped to
# the client attribute **token** and if the download method need it it will
# be passed automatically.
cli.secure_download(name='testfile.txt').save_all('/tmp')
```

More simple *direct value* mapping

```
from jsonwspclient import JsonWspClient

# direct param mapping: *token* param will be passed with value of '1234'.
cli = JsonWspClient('http://mysite.com', ['TransferService'], params_mapping={'token
↪': '1234'})
cli.secure_download(name='testfile.txt').save_all('/tmp')
```

```
from jsonwspclient import JsonWspClient
def get_token(method_name, **kwargs):
    """conditional token"""
    if method_name == 'get_user':
        return 'empty'
    return '12345'
cli = JsonWspClient(
    'http://mysite.com', ['Authenticate', 'TransferService'],
    params_mapping={'token': get_token})
cli.token = cli.get_user().result['token']
cli.donwload(name="testfile.txt").save_all('/tmp')
```

1.3.6 Fault handling

Simple fault handling by checking the `has_fault` property.

```
from jsonwspclient import JsonWspClient

with JsonWspClient('http://mysite.com', ['TransferService']) as cli:
    with cli.donwload(name='wrong-filename.txt') as res:
        if not res.has_fault:
            res.save_all('tmp')
```

Passing the `raise_for_fault` parameter to the service method.

```
from __future__ import print_function
from jsonwspclient import JsonWspClient
from jsonwspclient.jsonwspexceptions import JsonWspFault

with JsonWspClient('http://mysite.com', ['TransferService']) as cli:
    try:
        cli.donwload(raise_for_fault=True, name='wrong-filename.txt').save_all('/tmp')
    except JsonWspFault as error:
        print(error)
```

Passing the `raise_for_fault` parameter while instantiate the client.

```
from __future__ import print_function
from jsonwspclient import JsonWspClient
from jsonwspclient.jsonwspexceptions import JsonWspFault

with JsonWspClient('http://mysite.com', ['TransferService'], raise_for_fault=True) as␣
↪cli:
    try:
        cli.donwload(name='wrong-filename-1.txt').save_all('/tmp')
        cli.donwload(name='wrong-filename-2.txt').save_all('/tmp')
    except JsonWspFault as error:
        print(error)
```

Using the `raise_for_fault()` method.

```
from __future__ import print_function
from jsonwspclient import JsonWspClient
from jsonwspclient.jsonwspexceptions import JsonWspFault

with JsonWspClient('http://mysite.com', ['TransferService']) as cli:
    try:
        cli.donwload(name='wrong-filename-1.txt').raise_for_fault().save_all('/tmp')
        cli.donwload(name='wrong-filename-2.txt').raise_for_fault().save_all('/tmp')
    except JsonWspFault as error:
        print(error)
```

Warning: Remember while passing the `raise_for_fault=True` parameter, either to the service method or client creation, the *exception* will be raised **BEFORE** the *reponse processors* otherwise if you use the `raise_for_fault()` method you will need to take care about the *exceptions* in your *reponse processors*.

1.3.7 All together now (with subclassing)

```
from jsonwspclient import JsonWspClient
from jsonwspclient.jsonwspexceptions import JsonWspFault

# our event handler for file download monitoring.
def file_handler(event_name, value=0, max_value=0):
    """file Handler"""
    pct = value * float(max_value) / 100
    print("{} {}%\r".format(event_name, pct), end='')

# silly objectify function
def objectify(response, **dummy_kwargs):
    """objectify"""
    # our objpart will be an empty dict if response have some fault.
    # else it can be response.result.
    objpart = {} if response.has_fault else response.result
    # set the right objpart for the response.
    response.objpart = type('ObjPart', (object, ), objpart)
    # return the response.
    return response


# our client
class MyClient(JsonWspClient):
    """My Client"""
    # we can specify some thing in the class creation
    # we will download only so we will bind only the file.write event.
    events = [('file.read', file_handler)]
    # we will objectify the result.
    processors = [objectify]
    # and map the token param to the get_token method of the client.
    params_mapping = {'token': 'get_token'}
    user = None

    def authenticate(self, username, password):
        """Authenticate"""
        res = self.auth(username=username, password=password)
```

```
        # We set the user only if we not have faults.
        # (see the response processors).
        self.user = res.objpart if res.has_fault else None
        # Is a good practice to return the response if we are wrapping or
        # overriding some service method
        return res

    def get_token(self):
        """get token"""
        # return the user token (see params_mapping)
        return self.user.token

# instantiate the client.
with MyClient("http://mysite.com", ['Authenticate', 'TransferService']) as cli:
    # authenticate user.
    cli.authenticate('username', 'password')
    if cli.user:
        try:
            # try to download the file (automatically uses the user token as␣
↪parameter)
            # we use the :meth:`raise_for_fault` method which returns the response
            # or a JsonWspFault.
            cli.secure_download(name="testfile.txt").raise_for_fault().save_all("/tmp
↪")
        except JsonWspFault as error:
            print("error", error)
```

The example above can be write in a more simple way, but we need to mix features.

1.4 jsonwspclient

1.4.1 jsonwspclient package

Submodules

jsonwspclient.jsonwspclient module

Jsonwspclient `jsonwspclient.jsonwspclient`

class jsonwspclient.jsonwspclient.**JsonWspClient**(*url*, *services*, *headers=None*, *events=None*, *processors=None*, *params_mapping=None*, *raise_for_fault=False*, *auth=None*, *proxies=None*, *verify=False*, *response_class=None*, ***kwargs*)

Bases: `object`

JsonWsp Client.

The JSON-WSP Client class

Parameters

- **url** (*str*) – base url where to retrieve all services.
- **services** (*[str]*) – list of Service names to retrieve.

- **headers** (*dict*) – Headers to add or repalce.
- **events** (*[(str, function)]*) – list of tuples contaning the event name and the relative function.
- **processors** (*[function]*) – list of functions that can process and/or modify responses before they are returned.
- **params_mapping** (*dict*) – Dictionary with mapping for client attributes or methods to service command parmaters.
- **raise_for_fault** (*bool*) – Automatically raise Exceptions on JSON-WSP response faults.
- **auth** (*any*) – Authentication according with Requests Authentication in most case a simple tuple with **username** and **password** should be enough.
- **proxies** (*dict*) – Dictionary mapping protocol to the URL of the proxy (see Requests proxies).
- **verify** (*bool, str*) – Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. (see Requests SSL Cert Verification).
- **response_class** (*JsonWspResponse subclass*) – Custom Response class wich subclass JsonWspResponse (default JsonWspResponse).

add_events(**events*)
Add events.

close()
Close.

events = []
([(str, function)]) – list of tuples contaning the event name and the relative function.

method
return method.

Parameters **name** (*str*) – name of the service to retrieve

Returns the services method if possible.

Return type function

params_mapping = {}
(dict) – Dictionary with mapping for client attributes or methods to service command parmaters.

post(*path*, *data=None*, *method='POST'*)
Post a request.

Parameters

- **path** (*str*) – Path relative to base url of the client instance.
- **data** (*dict*) – Dictionary with data to post (will be convert into json string).
- **method** (*str*) – Method to use (default to POST)

Returns The response to the request.

Return type *JsonWspResponse*

post_mp(*path*, *data=None*, *attachs=None*, *method='POST'*)
Post a multipart requests.

Parameters

- **path** (*str*) – Path relative to base url of the client instance.
- **data** (*dict*) – Dictionary with data to post (will be convert into json string).
- **attachs** (*dict*) – Dictionary with files id and relative file object. ({fileid: fileobject})
- **method** (*str*) – Method to use (default to POST)

Returns The response to the request.

Return type *JsonWspResponse*

processors = []
([function]) – list of functions that can process and/or modify responses before they are returned.

remove_events(**events*)
Remove events.

service
return service.

Parameters **name** (*str*) – name of the service to retrieve

Returns the service object

Return type *JsonWspService*

jsonwspclient.jsonwspexceptions module

Jsonwspexceptions jsonwspclient.jsonwspexceptions

exception jsonwspclient.jsonwspexceptions.**ClientFault**(**args*, ***kwargs*)
Bases: *jsonwspclient.jsonwspexceptions.JsonWspFault*

Client Fault.

exception jsonwspclient.jsonwspexceptions.**IncompatibleFault**(**args*, ***kwargs*)
Bases: *jsonwspclient.jsonwspexceptions.JsonWspFault*

Incompatible Fault.

exception jsonwspclient.jsonwspexceptions.**JsonWspException**
Bases: exceptions.Exception

Base Exception

exception jsonwspclient.jsonwspexceptions.**JsonWspFault**(**args*, ***kwargs*)
Bases: *jsonwspclient.jsonwspexceptions.JsonWspException*

Base exception.

code = ''

description = ''

details = ()

fault = {}

filename = ()

hint = ''

lineno = ()

exception jsonwspclient.jsonwspexceptions.**ParamsError**
Bases: *jsonwspclient.jsonwspexceptions.JsonWspException*

Params Errror.

exception jsonwspclient.jsonwspexceptions.**ServerFault**(**args*, ***kwargs*)
Bases: *jsonwspclient.jsonwspexceptions.JsonWspFault*

Server fault error.

jsonwspclient.jsonwspmultipart module

Jsonwspmultipart jsonwspclient.jsonwspmultipart

class jsonwspclient.jsonwspmultipart.**JsonWspAttachment**(*index=0*)
Bases: object

Class for the attachments

Parameters **index** (*int*) – Attachment index.

descriptor
any – File descriptor.

path
str – Temporary file path.

att_id = None
(str) – Attachment id.

close()
Try to close the temp file.

filename = None
(str) – filename if found in headers.

headers = None
(CaseInsensitiveDict) – attachment headers.

index = None
(int) – Attachment index.

open(*mode='rb'*)
Open the temp file and return the opened file object

Parameters **mode** (*srt, optional*) – open mode for the file object.

Returns the open file.

Return type (file)

save(*path*, *filename=None*, *overwrite=True*)
Save the file to path

Parameters

- **path** (*str*) – Path where to save the file.
- **filename** (*str, optional*) – Name for the file (if not already in path)
- **overwrite** (*bool, optional*) – Overwrite the file or no (default True)

Raises ValueError – if a filename is not found.

Note: If *path* is just a folder without the filename and no filename param is specified it will try to use the filename in the content-disposition header if one.

size = None
(int) – Attachment size.

update(*headers*)
update headers

class jsonwspclient.jsonwspmultipart.**JsonWspAttachmentMeta**
Bases: type

Meta for instance check

class jsonwspclient.jsonwspmultipart.**MultiPartReader**(*headers*, *content*, *size=None*, *chunk_size=8192*)
Bases: object

Reader

iterator(*chunk_size=None*)
Iterator

read(*chunk_size=None*)

read_all(*chunk_size=None*)

read_chunk(*size=None*)

write(*data*, *save=False*)
Write

class jsonwspclient.jsonwspmultipart.**MultiPartWriter**(*jsonpart*, *files*, *chunk_size=8192*, *boundary=None*, *encoding='UTF-8'*)
Bases: object

close()
Close

next()
Next

read(*chunk_size=None*)
Read

jsonwspclient.jsonwspmultipart.**get_filename**()
findall(string[, pos[, endpos]]) –> list. Return a list of all non-overlapping matches of pattern in string.

jsonwspclient.jsonwspmultipart.**get_headers**()
findall(string[, pos[, endpos]]) –> list. Return a list of all non-overlapping matches of pattern in string.

jsonwspclient.jsonwspmultipart.**split_headers**()
search(string[, pos[, endpos]]) –> match object or None. Scan through string looking for a match, and return a corresponding match object instance. Return None if no position in the string matches.

jsonwspclient.jsonwspmultipart.**stringify_headers**(*headers*, *encoding='UTF-8'*)
stringi

jsonwspclient.jsonwspresponse module

Jsonwspresponse jsonwspclient.jsonwspresponse

class jsonwspclient.jsonwspresponse.**JsonWspResponse**(*response*, *trigger*)
Bases: object

JsonWspResponse (wrapper for requests Response object) is not meant to be instantiate manually but only as response from *JsonWspClient* requests.

attachments = None
(dict) – Attachments dictionary, not really useful.

fault = None
(dict) – Fault dictionary if response has fault.

fault_code = None
(str) – Fault code if response has fault.

has_fault = None
(bool) – True if response has fault.

is_multipart = None
(bool) – True if response is multipart.

length = None
(int) – response content length

next()
If JsonWspResponse is multipart returns the next attachment.

Returns the attachment object.

Return type *JsonWspAttachment*

raise_for_fault()
Reise error if needed else return self.

read_all(*chunk_size=None*)
Read all the data and return a Dictionary containig the Attachments.

Parameters **chunk_size** (*int*) – bytes to read each time.

Returns Dictionary with all attachments.

Return type dict

response_dict = None
(dict) – JSON part of the response.

result = None
(dict,list) – **data** of the JSON part of the response.

save_all(*path*, *name='name'*, *overwrite=True*)
Save all the attachments ad once.

Parameters

- **path** (*str*) – Path where to save.
- **name** (*str, optional*) – key with which the file name is specified in the dictionary (default `name`).
- **overwrite** (*bool, optional*) – overwrite the file if exists (defautl True).

jsonwspclient.jsonwspservice module

Jsonwspservice jsonwspclient.jsonwspservice

class jsonwspclient.jsonwspservice.**JsonWspService**(*client*, *service_name*)
Bases: object

Service.

list_methods()
list_methods.

jsonwspclient.jsonwsputils module

Jsonwsputils jsonwspclient.jsonwsputils

class jsonwspclient.jsonwsputils.**FileWithCallBack**(*path*, *callback*, *mode='rb'*, *size=0*)
Bases: object

FileWithCallBack.

close()
Close.

read(*size*)
read.

write(*data*)
write.

class jsonwspclient.jsonwsputils.**Observer**(*events*)
Bases: object

Observer for events.

add(*name*, *funct*)
add event.

remove(*name*, *funct*)
remove event.

trigger(*event*, ***kwargs*)
Trigger.

jsonwspclient.jsonwsputils.**check_attachment**(*items*)
check_attachment.

jsonwspclient.jsonwsputils.**fix_attachment**(*val*, *attachment_map*)
Fix attachment.

jsonwspclient.jsonwsputils.**get_boundary**(*headers*)
return boundary.

jsonwspclient.jsonwsputils.**get_charset**(*headers*)
return charset.

jsonwspclient.jsonwsputils.**get_fileitem**(*path*, *data='data'*, *name='name'*, *mode='rb'*)
get fileitem.

jsonwspclient.jsonwsputils.**get_multipart**(*headers*)
return multipart.

jsonwspclient.jsonwsputils.**has_attachments**()
match(string[, pos[, endpos]]) –> match object or None. Matches zero or more characters at the beginning of the string

jsonwspclient.jsonwsputils.**make_method**(*funct*, *instance*, *cls*)
Make method

jsonwspclient.jsonwsputils.**walk_args_dict**(*kwargs*, *attachment_map*)
Walk args.

Module contents

__init__ jsonwspclient.__init__

1.5 Changelog

1.5.1 Version 0.0.1

- Feature A added

1.6 License

```
The MIT License (MIT)

Copyright (c) 2018 ellethee

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

1.7 Developers

- ellethee <luca800@gmail.com>

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index