python-gtmetrix2

Alexey Shpakovsky

Sep 14, 2021

CONTENTS:

Warning: As long as major version of this library is 0 (i.e. library has version 0.x.y), API is not guaranteed to be compatible between versions. When you start using this library, **please** let the developer know about - I will bump the major version to 1, and usuall semver guatantees regarding version compatibility will be applied.

python-gtmetrix2 is a Python client library for GTmetrix REST API v2.0 (hence 2 in the name).

Inspired by the library with a similar name.

CHAPTER

ONE

INSTALLATION:

1.1 via pip

```
pip install python-gtmetrix2
```

1.2 manual

- Clone or download and extract the repository
- Copy the `src/python-gtmetrix2` directory to your project directory

CHAPTER

TWO

USAGE:

Simplest example:

```
import json
import python_gtmetrix2

api_key = "e8ddc55d93eb0e8281b255ea236dcc4f"    # your API key
url = "http://example.com"                      # URL to test

account = python_gtmetrix2.Account(api_key)     # init
test = account.start_test(url)                  # start test
test.fetch(wait_for_completion=True)            # wait for it to finish
report = test.getreport()                       # get test result

print(json.dumps(report, indent=2))             # do something useful with it
```

For a wordy introduction into this library, please see next chapter. For more advanced examples, see the examples section. For a more technical explanation, dive into API reference.

2.1 Introduction

Warning: As long as major version of this library is 0 (i.e. library has version 0.x.y), API is not guaranteed to be compatible between versions. When you start using this library, **please** let the developer know about - I will bump the major version to 1, and usuall semver guatantees regarding version compatibility will be applied.

2.1.1 Account

Main entry point for this library is *Account* class which is initialized like this:

```
account = python_gtmetrix2.Account(api_key)
```

where `api_key` is your GTmetrix API key.

Object of this class lets you start tests, like this:

```
test = account.start_test(url)
```

where `url` is the url you want to test. Optionally, you can pass extra arguments, like this:

```
test = account.start_test(url, report='none', adblock=1)
```

Full list of available parameters is available in GTmetrix API documentation, section "Test Parameters". This call returns an object of type *Test*. Note that this call does **not** wait for the test to finish. To know how to wait for the test to finish, read on.

2.1.2 Test

Object of type *Test* has two methods which you will be using: *fetch()* and *getreport()*. Method *fetch()* updates test information from GTmetrix API server and has an optional argument `wait_for_completion`, which, when set to `True`, instructs this method to wait until the test finishes.

If the test completes successfully (which happens most of the time), you can use *getreport()* method to retrieve test results in the form of *Report* object, like this:

```
test.fetch(wait_for_completion=True)
report = test.getreport()
```

Note that `report` might be `None` if test did not finish successfully (for example, due to connection or certificate error).

2.1.3 Report

Report is a descendant of `dict`, so you can treat it like one:

```
print(json.dumps(report, indent=2))
```

Report also has *getresource* method which lets you save a report resource (like a PDF representation of the report, screenshot, or a video of loading website) to file or a variable in your program:

```
report.getresource('report.pdf', 'report.pdf')
```

That's all for now. More examples can be found in examples directory in the repo.

2.2 Examples

Warning: As long as major version of this library is 0 (i.e. library has version 0.x.y), API is not guaranteed to be compatible between versions. When you start using this library, **please** let the developer know about - I will bump the major version to 1, and usuall semver guatantees regarding version compatibility will be applied.

2.2.1 Simplest example

You already saw it on the main page:

```
import json
import python_gtmetrix2

api_key = "e8ddc55d93eb0e8281b255ea236dcc4f"    # your API key
```

(continues on next page)

(continued from previous page)

```
url = "http://example.com"                          # URL to test

account = python_gtmetrix2.Account(api_key)         # init
test = account.start_test(url)                      # start test
test.fetch(wait_for_completion=True)                # wait for it to finish
report = test.getreport()                           # get test result

print(json.dumps(report, indent=2))                 # do something useful with it
```

All the following examples are available in the github repo in "examples" subdir

On this page, only the most interesting part is shown - without `include` statements and `if __name__ == "__main__"` part which makes them into executable scripts.

2.2.2 Start multiple tests

Example which shows how to start multiple tests in parallel, wait for them to finish, and fetch reports for tests that completed successfully.

Note that GTmetrix limits the number of tests you can run in parallel (2 concurrent tests on a Basic account, 8 concurrent tests on a PRO account). This example does not impose any concurrency limits by itself, but instead relies on GTmetrix API to reply with 429 HTTP error and retries.

```
def main(api_key, urls):
    account = python_gtmetrix2.Account(api_key)

    print("=== starting tests ===")

    tests = []
    for url in urls:
        test = account.start_test(url)
        print(json.dumps(test))
        tests.append(test)

    print("=== wait for tests to finish ===")

    for test in tests:
        test.fetch(wait_for_completion=True)

    print("=== fetching report for each test ===")

    for test in tests:

        report = test.getreport()

        if report is None:
            print("No report for test %s" % test["id"])
        else:
            print(json.dumps(report, indent=2))
```

2.2.3 List recent tests

Example which shows what can be done with result of *Account.list_tests()* method.

You can either treat it as a JSON-like *dict* object, or use *Test.getreport()* function to get corresponding report, *if it exists*. *Report* is also a JSON-like *dict* object.

```
def main(api_key):
    account = python_gtmetrix2.Account(api_key)

    print("=== fetching tests ===")

    tests = account.list_tests()

    if len(tests) == 0:
        print("No tests found! Note that only tests started within last 24 hours are␣
↪available via this API.")
        return

    for test in tests:
        print(json.dumps(test, indent=2))

    print("=== fetching report for each test ===")

    for test in tests:

        report = test.getreport()

        if report is None:
            print("No report for test %s" % test["id"])
        else:
            print(json.dumps(report, indent=2))
```

2.2.4 Operations on report

Example which shows some possible uses of report:

- You can treat it as a JSON-like *dict* object and access any properties you want
- You can request report to be deleted or retested
- You can download a single report resource (such as a PDF version)

Also, this example demonstrates how you can work with JSON resources, like a har file.

Note how it uses *Account.reportFromId()* method to get report from its ID. When running examples, you can get report ID from "links.report" attribute of the test object (note that it points to the whole report URL, and the report ID is the part which comes after */reports/* part), or from report's *id* attribute.

When using this library, you can also use *Test.getreport()* method to get report object for a specific test object.

```
def main(api_key, report_id, operation="print", *args):
    """Usage: %s api_key report_id [operation]
    or: %s api_key report_id getresource resource [filename]

    where operation is one of: print (default), delete, retest, size, getresource
```

(continues on next page)

(continued from previous page)

```
    getresource operation requires one extra argument: what resource to get,
    and one optional: filename where to save it. If filename is not provided,
    resource is printed to stdout.
    """

    account = python_gtmetrix2.Account(api_key)
    report = account.reportFromId(report_id)

    if operation == "print":
        print(json.dumps(report, indent=2))
        # print(report["attributes"]["first_contentful_paint"])

    elif operation == "delete":
        report.delete()
        print("Report deleted.")

    elif operation == "retest":
        test = report.retest()
        print("new test:")
        print(json.dumps(test, indent=2))

    elif operation == "getresource":
        if len(args) not in [1, 2]:
            print("Usage: %s api_key report_id getresource resource [filename]" % sys.
↪argv[0])
            print("If filename is not provided, resource is printed to stdout.")
            exit()
        getresource(report, *args)

    elif operation == "size":
        har = json.loads(report.getresource("net.har").decode())
        size_bytes = summarizeHar(har)
        size_kb = size_bytes / 1024
        size_mb = size_kb / 1024
        print("Total size of all resources, uncompressed: %d bytes = %.1f kb = %.1f MB"
↪% (size_bytes, size_kb, size_mb))

    else:
        print("Usage: %s api_key report_id [operation]" % sys.argv[0])
        print("or: %s api_key report_id getresource resource [filename]" % sys.argv[0])
        print("where operation is one of: print (default), delete, retest, size,␣
↪getresource")


def getresource(report, resource, filename=sys.stdout.buffer):
    """Gets report resource and saves it to filename (stdout by default)"""
    report.getresource(resource, filename)


def summarizeHar(har):
    """Given a har file (parsed json object), returns total size of all responses, in␣
↪bytes."""
```

(continues on next page)

(continued from previous page)

```
    return sum((entry["response"]["content"]["size"] for entry in har["log"]["entries"]))
```

2.3 API reference

Warning: As long as major version of this library is 0 (i.e. library has version 0.x.y), API is not guaranteed to be compatible between versions. When you start using this library, **please** let the developer know about - I will bump the major version to 1, and usuall semver guatantees regarding version compatibility will be applied.

2.3.1 Overview

User of this library is expected to interact primarily with the following three classes:

- *Account*, which is instantiated with your API key and is used for all API calls which don't operate on a particular test or report. For example, API calls to start a *new* test (*Account.start_test()*), or to get account information (*Account.status()*).
- *Test*, which corresponds to a requested test (which might be still running or already finished).
- *Report*, which describes results of a *successfully finished* test.

Note that usually objects of *Test* and *Report* classes should not be instantiated directly - users of this library are expected to use methods of *Account* class instead: for example, *Account.start_test()* to start a test, or *Account.list_tests()* to get a list of recent tests. And then *Test.getreport()* to get a report for a finished test.

Also note that *Test* and *Report* classes are descendants of the dict, so you can operate on as such: `json.dumps()` them to inspect their internals, and access their attributes same way as for a `dict`.

2.3.2 Public API classes

class python_gtmetrix2.**Account**(*api_key*, *base_url='https://gtmetrix.com/api/2.0/'*, *sleep_function=<built-in function sleep>*)

Bases: `object`

Main entry point into this library

Parameters

- **api_key** (`str`) – your GTmetrix API key.
- **base_url** (*str, optional*) – base URL for all API requests - useful for testing or if someone implements a GTmetrix competitor with a compatible API, defaults to "https://gtmetrix.com/api/2.0/"
- **sleep_function** (*method, optional*) – the function to execute when waiting between retries (after receiving a "429" response) - useful for testing, or if someone wants to add some logging or notification of a delayed request, defaults to `time.sleep()`

start_test(*url*, ***attributes*)

Start a Test

Parameters **url** (*str*) – the URL to test.

You can pass additional parameters for the tests (like browser, location, desired report depth, etc) as extra keyword arguments, like this:

```
>>> account.start_test('http://example.com', report='none')
```

Or, if you prefer having a dict, you can use the **kwargs-style Python expansion, like this:

```
>>> parameters={'location': '1', 'browser': '3', 'adblock': '1'}
>>> account.start_test('http://example.com', **parameters)
```

Note that this method does not wait for the test to finish. For that, call *test.fetch(wait_for_completion=True)* after calling this method.

Returns a new instance of *Test* corresponding to a new running test.

Return type *Test*

list_tests(*sort=None*, *filter=None*, *page_number=0*)

Get a list of recent tests.

Note that while *reports* are stored on GTmetrix servers for several (1 to 6) months, tests are deleted after 24 hours. Hence, this function lists only rather recent tests.

Parameters

- **sort** (*str, optional*) – Sort string by one of "created", "started", "finished", optionally prefixed with "-" for reverse sorting, defaults to None (no sort).
- **filter** (*dict*) – Filter tests - argument should be a dict of key/value pairs, where key is one of "state", "created", "started", "finished", "browser", "location", optionally postfixed with one of ":eq, :lt, :lte, :gt, :gte" and value is, well, value (string or number). Valid values for "state" are "queued", "started", "error", and "completed". "created", "started" and "finished" are UNIX timestamps. "browser" and "location" are browser and location IDs.

Return type list(*Test*)

Examples To get all tests finished successfully within last 10 minutes:

```
>>> import time
>>> now = int(time.time())
>>> tests = account.list_tests(
...     filter={"state": "completed", "created:gt": (now-10*60)})
```

To get all tests which ended up with an error, and print the error message for each of them:

```
>>> tests = account.list_tests(filter={"state": "error"})
>>> for test in tests:
...     print("Test %s failed: %s" % (test["id"], test["attributes"][
↪"error"]))
```

status()

Get the current account details and status.

Returns dict with information about your api key, current API credit balance, and time of next credit refill (Unix timestamp).

Example

```
>>> account = Account("e8ddc55d93eb0e8281b255ea236dcc4f")
>>> status = account.status()
>>> print(json.dumps(status, indent=2))
```

would print something like this:

```
{
  "type": "user",
  "id":   "e8ddc55d93eb0e8281b255ea236dcc4f",
  "attributes": {
    "api_credits": 1497,
    "api_refill":  1618437519
  }
}
```

testFromId(*test_id*)
Fetches a test with given id and returns the corresponding *Test* object.

Parameters **test_id** (*str*) – ID of test to fetch. Note that if such test does not exist, an exception will be raised.

Return type *Test*

reportFromId(*report_id*)
Fetches a report with given id and returns the corresponding *Report* object.

Parameters **report_id** (*str*) – ID (slug) of report to fetch. Note that if such report does not exist, an exception will be raised.

Return type *Report*

class python_gtmetrix2.**Test**(*requestor*, *data*, *sleep_function=<built-in function sleep>*)
Bases: *python_gtmetrix2._internals.Object*

fetch(*wait_for_completion=False*, *retries=10*)
Ask API server for updated data regarding this test.

Parameters

- **wait_for_completion** (*bool, optional*) – Whether to wait until the test is finished, defaults to False
- **retries** (*int, optional*) – Number of retries before giving up, defaults to 10

getreport()
Returns Report object for this test, if it is available.

Note that this function does not *check* whether the test has actually completed since the last call to API. For that, you should use method *fetch* first.

Also note that even if report is *finished* (i.e. after *fetch(wait_for_completion=True)* returns), it's not guaranteed that it *completed successfully* - it could have finished with an error - for example, due to certificate or connection error. In that case, your test will have *status = "error"* attribute, and also *error* attribute explaining what went wrong.

Return type *Report* or None

class python_gtmetrix2.**Report**(*requestor*, *data*, *sleep_function=<built-in function sleep>*)
Bases: *python_gtmetrix2._internals.Object*

delete()
Delete the report.

Note that after executing this method, all other methods should error with a "404 Report not found" error.

retest()
Retest the report.

Returns a new instance of *Test* corresponding to a new running test.

Return type *Test*

`getresource`(*name*, *destination=None*)

Get a report resource (such as a PDF file, video, etc)

Depending on the value of `destination` parameter, it might be saved to a file, a file-like object, or returned to the caller. Be careful with the latter in case a file is too big, though.

Parameters

- **name** (*str*) – Name of the desired resource. You can find full list at the GTmetrix API documentation: <https://gtmetrix.com/api/docs/2.0/#api-report-resource>
- **destination** (*None or str or a file-like object*) – Where to save the downloaded resource. If it is `None`, then resource is completely downloaded into RAM and returned to the caller. If it is a string, then the resource is saved into a file with that name. If it is a file-like object, then `shutil.copyfileobj()` is used to copy the resource into that object.

2.3.3 exceptions

Warning: As long as major version of this library is 0 (i.e. library has version 0.x.y), API is not guaranteed to be compatible between versions. When you start using this library, **please** let the developer know about - I will bump the major version to 1, and usuall semver guatantees regarding version compatibility will be applied.

Overview

Basically, there are two main exception classes: *APIFailureException* and *APIErrorException*.

First of them (the "failure" one) happens when API server returns something what was not expected by the library: for example, when library expects to receive a JSON, but can't parse the response. Cases like this should not happen outside of unittests, so if you encounter one - please file an issue.

Second one (the "error" one) happens when API server returns (properly formatted) error response. In that case, it is assumed that it was a problem with how the library is used. But if you disagree - please file an issue.

Both of these classes are based on the *BaseAPIException*, and has the following attributes usually set: `request`, `response`, `data`. `request` and `response` link to relevant instances of `urllib.request.Request`, `http.client.HTTPResponse`, or `urllib.error.HTTPError`, if they were available at the moment when the exception was raised. In addition to this, *APIFailureException* has a `message` attribute, which contains a text description of the problem.

Also, there is a *APIErrorFailureException*, which is raised if *APIFailureException* (i.e. unparsable or invalid JSON) happens. It's a subclass of *APIFailureException*, so you don't need to care about it, unless you're interested in it.

Reference

exception python_gtmetrix2.exceptions.**BaseAPIException**(*request*, *response*, *data*, *extra=None*)
Bases: Exception

Base class for all exceptions in this library. Passed parameter are available as attributes.

Parameters

- **request** (urllib.request.Request or None) – Request which was sent to the API server, if available.
- **response** (http.client.HTTPResponse or urllib.error.HTTPError) – Response from the API server.
- **data** (*None*, *bytes* or *dict* (*in case it was parsed from JSON*)) – data received from the API server, if any.
- **extra** – extra information, if available, defaults to None.

exception python_gtmetrix2.exceptions.**APIFailureException**(*message*, **args*)
Bases: *python_gtmetrix2.exceptions.BaseAPIException*

API server returned an unexpected response.

There was a disagreement between API server and this library: server returned something what the library did not expect to receive.

Parameters **message** (*str*) – text explaining the error.

other parameters are same as for parent class *BaseAPIException*.

exception python_gtmetrix2.exceptions.**APIErrorFailureException**(*message*, **args*)
Bases: *python_gtmetrix2.exceptions.APIFailureException*

APIFailureException happened when processing an error response.

Parameters are the same as for parent class *APIFailureException*.

exception python_gtmetrix2.exceptions.**APIErrorException**(*request*, *response*, *data*, *extra=None*)
Bases: *python_gtmetrix2.exceptions.BaseAPIException*

API returned an error.

Parameters are the same as for parent class *BaseAPIException*.

You can inspect error details in the *data* attribute of this object, it usually looks like this:

```
{
  "errors": [
    {
      "status": "405",
      "code": "E40500",
      "title": "HTTP method not allowed",
      "detail": "Method is not supported by the endpoint"
    }
  ]
}
```

2.3.4 internal details

Warning: Contents of this module considered to be internal and therefore may change without warning. If your code uses something in this module, please let the developer know about it so we could consider adding it to public API.

Overview

All requests are made by the instance of *Requestor* which is created by the `Account` class and is usually shared between all instances created from it.

It also uses *NoRedirect* to avoid redirections when API returns both 30x redirect code and actual data in response body. It is important, for example, when requesting data for a finished test (`Test.fetch()`) - in that case, we would like to store received data in the `Test` object, but by default Python will throw away received data and follow redirect.

There are four helper functions to check if received JSON represents a valid *error*, *test*, *report*, *user*.

`Test` and `Report` classes have a same parent class *Object*, which holds elements that are common to both of them (basically, constructor and parent class)

Reference

class python_gtmetrix2._internals.**NoRedirect**

Bases: urllib.request.HTTPRedirectHandler

Helper class for avoiding redirection on 30x responses. From https://stackoverflow.com/a/52086806

redirect_request(*req*, *fp*, *code*, *msg*, *headers*, *newurl*)

Returns None to avoid redirection.

class python_gtmetrix2._internals.**Requestor**(*api_key*, *base_url='https://gtmetrix.com/api/2.0/'*, *sleep_function=<built-in function sleep>*)

Bases: object

Class for making requests.

It also manages authentication, optionally follows redirects, and retries on "429" responses.

Note that usually objects of this class should not be instantiated directly - you can use methods of `Account` class instead.

Parameters are the same as for `Account`

request(*url*, *follow_redirects=False*, *data=None*, ***kwargs*)

Make a request and return the response.

Parameters

- **url** (*str*) – URL to request (base URL will be prepended)
- **follow_redirects** (*bool, optional*) – Whether to follow 30x redirects, defaults to False.
- **data** (*dict (to be JSON-encoded), string or bytes, optional*) – data to send as request body (usually with a POST request), defaults to None
- **method** (*str, optional*) – method to use for request ("GET", "POST", etc.), defaults to None to let urllib to decide (POST if data is provided, GET otherwise)

- **headers** (*dict, optional*) – headers to send with the request, in format understood by urllib, defaults to {}
- **retries** (*int, optional*) – Number of times to retry on "429 Rate limit exceeded" responses, defaults to 10
- **return_data** (*bool, optional*) – whether this function should read() the response, parse it as JSON, validate it (check that it's a dict and has a "data" key), and return that JSON - **or** if it should let the caller deal with it. Latter is useful for API calls which return files instead of JSON.

Returns Tuple of 2 elements: response object and response data. When API returns HTTP status code in [200..299] range, "response object" is an instance of `http.client.HTTPResponse`. However, when API returns code 30x, Python considers it an error, so "response object" is an instance of `urllib.error.HTTPError` instead. Second element of the returned tuple is parsed JSON (`dict`), *unless return_data* parameter was *False*. In latter case, it's responsibility of the caller to call *read* method on the response object (conveniently, both types of returned objects support it).

Return type tuple(http.client.HTTPResponse or urllib.error.HTTPError, dict or None)

python_gtmetrix2._internals.**dict_is_error**(*data*)
helper function to check whether passed argument is a proper `dict` object describing an error.

Parameters **data** (*dict*) – value to check

Return type bool

python_gtmetrix2._internals.**dict_is_test**(*data*)
helper function to check whether passed argument is a proper `dict` object describing a test.

Parameters **data** (*dict*) – value to check

Return type bool

python_gtmetrix2._internals.**dict_is_report**(*data*)
helper function to check whether passed argument is a proper `dict` object describing a report.

Parameters **data** (*dict*) – value to check

Return type bool

python_gtmetrix2._internals.**dict_is_user**(*data*)
helper function to check whether passed argument is a proper `dict` object describing a user.

Parameters **data** (*dict*) – value to check

Return type bool

class python_gtmetrix2._internals.**Object**(*requestor*, *data*, *sleep_function=<built-in function sleep>*)
Bases: `dict`

Base class for `Test` and `Report` classes.

Note that usually objects of these classes should not be instantiated directly - you can use methods of `Account` class instead.

Also note that since they are descendants of the `dict`, you can simply `json.dumps()` them to inspect their internals.

Parameters

- **requestor** (`Requestor`) – Requestor object to use for requests made by this object

- **data** (*dict*) – initial data. Note that it is responsibility of the caller to ensure that it contains valid data (passes respective *dict_is_** check).
- **sleep_function** (*method, optional*) – the function to execute when waiting between retries (after receiving a "429" response) - useful for testing, or if someone wants to add some logging or notification of a delayed request, defaults to `time.sleep()`

2.4 Contributing

Any feedback or PRs are welcome

Note: Saying that, I would greatly appreciate if you drop me a line (or open guthub issue) before making changes - so we're on the same page and don't do similar or incompatible changes.

2.4.1 Testing

Autotests are automated by Travis in clouds, so to run them you can just create a PR.

To run tests locally, you need to install pytest with httpserver and execute `pytest` in the root of this repository like this:

```
~/git/python-gtmetrix2$ pytest tests
======================== test session starts ========================
platform linux -- Python 3.9.6, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /home/lex/git/python-gtmetrix2
plugins: requests-mock-1.9.3, cov-2.12.1, httpserver-1.0.0
collected 15 items

tests/auto_test.py ...............                                 [100%]

========================= 15 passed in 0.80s =========================
```

Examples serve as kind of "manual" tests.

2.4.2 Coverage

Code coverage by autotests is measured by Codecov in clouds, so you can see results in PRs.

To measure coverage manually, install coverage and run it like this:

```
~/git/python-gtmetrix2$ coverage run -m pytest tests
```

Its output is same as when running tests. To show actual coverage values, run:

```
~/git/python-gtmetrix2$ coverage report --skip-empty
Name                                Stmts   Miss  Cover
-----------------------------------------------------
src/python_gtmetrix2/__init__.py      155      0   100%
tests/auto_test.py                    227      0   100%
-----------------------------------------------------
TOTAL                                 382      0   100%
```

To generate a coverage report in html format, run:

```
~/git/python-gtmetrix2$ coverage html --skip-empty
```

It will output nothing, but create nice HTML report in the `htmlcov` directory.

Tis project aims for 100% code coverage by tests, so just mark untested lines with `pragma: no cover` and be done with it, lol.

2.5 Recent changes

Note: According to semver, as long as major version is 0, no guarantees regarding version compatibility are given. Any breaking changes are possible. When this library sees a first user, its major version will be bumped to 1.

2.5.1 0.3.0

- library wa split into modules for easier differentiation between public and private parts
- Docs: better examples exposure and overview.

2.5.2 0.2.0

- Argument for the *Test.fetch()* was renamed from `wait_for_complete` to `wait_for_completion`
- Bugfix: *Test.fetch()* was following redirects when it wasn't supposed to. Due to this, it was fetching "report" JSON and crashed.

2.5.3 0.1.2

- Class `Interface` was renamed to *Account*
- "GTmetrix" name was removed from exception names
- Docs: separate pages for changelog and contributing.

2.5.4 0.1.1

- Added documentation
- Added *Interface.testFromId* and *reportFromId* methods
- Changed the way how errors are checked - now `Requestor._plain_request()` respect HTTP status instead of parsing the response JSON.
- Changed Travis config to deploy to both "test" and "main" pypi's

2.5.5 0.1.0

Initial release

2.5.6 0.0.x

Non-release versions (currently published to test.pypi.org). Last number in version is build number in Travis and increases over time. Hence, recent 0.0.x versions might be "newer" than some 0.1.x versions!

2.6 Index

This is a hack to add auto-generated index to the TOC.

PYTHON MODULE INDEX

p

INDEX