whispyr Documentation

Release 0.3.0

Grigory Starinkin

Aug 14, 2018

Contents:

CHAPTER 1

whispyr

a python client library for whispir.io

- Free software: MIT license
- Documentation: https://whispyr.readthedocs.io.

1.1 Features

- Very thin client for *whispir.io* API.
- Adding a new whispir endpoint as easy as declaring a new class (almost)
- Configurable retry policies
- Decent test code coverage

CHAPTER 2

Installation

2.1 Stable release

To install whispyr, run this command in your terminal:

```
$ pip install whispyr
```

This is the preferred method to install whispyr, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for whispyr can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/velimir0xff/whispyr
```

Or download the tarball:

```
$ curl  -OL https://github.com/velimir0xff/whispyr/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

3.1 Initialisation

To use whispyr in a project you need to have valid whispir.io credentials (username, password and API key):

```
from whispyr import Whispir

TEST_USERNAME = 'U53RN4M3'
TEST_PASSWORD = 'P4ZZW0RD'
TEST_API_KEY = 'V4L1D4P1K3Y'

whispir = Whispir(TEST_USERNAME, TEST_PASSWORD, TEST_API_KEY)
```

3.2 Available resources

Instance of class `Whispir` contains all available resources (workspaces, messages, templates, response rules and etc, you can find more on official whispir API documentaion page https://whispir.github.io/api) as its members:

```
>>> whispir.workspaces
<whispyr.whispyr.Workspaces object at 0x10f1c2eb8>
>>> whispir.messages
<whispyr.whispyr.Messages object at 0x10f1c2e80>
...
```

Resources in plural form are collections which provide an access to containers. For example `workspaces` is a workspace resource and it provides CRUD functionality for workspaces.

3.3 Collections actions

Each collection has `create`, `show`, `list`, `update` and `delete` functions to work with objects. All passed arguments passed as is to json encoder. Library doesn't perform any checks or validations, which means that you need to know required parameters and their format before you can call any of mentioned actions.

3.3.1 create

`create` makes a `POST` request for a given resource:

```
name = 'whispyr tests'
new_workspace = whispir.workspaces.create(projectName=name, status='A')
```

returned object is a container for a created resource. For this example it's an instance of `whispyr.Workspace`:

```
from whispyr import Workspace
assert isinstance(workspace, Workspace)
```

All containers are dictionary like objects which wraps response from whispir.io API and provides an access to other resources available under some particular instance of an entity.

3.3.2 show

`show` makes a `GET` request for a given resource with a provided ID:

```
workspace = whispir.workspaces.show(new_workspace['id'])
assert isinstance(workspace, Workspace)
assert 'id' in workspace
assert 'projectName' in workspace
```

3.3.3 list

`list` makes a `GET` request for a given resource to return all available objects (library takes care about pagination):

```
for workspace in whispir.workspaces.list():
    assert isinstance(workspace, Workspace)
    assert 'id' in workspace
    assert 'projectName' in workspace
```

3.3.4 update

`update` makes a `PUT` request for a given resource with a provided ID to update/edit a resource with specified parameters:

```
whispir.contacts.update(
    contact_id,
    firstName='John',
    lastName='Wick',
    timezone='Australia/Melbourne',
    workCountry='Australia'
)
```

3.3.5 delete

`delete` makes a `DELETE` request for a given resource with a provided ID:

```
whispir.contacts.delete(contact['id'])
```

3.4 Container resources

Some containers (such as `whispyr.Workspace` and `whispyr.Message`) might provide an access to some collections.

3.4.1 Workspace

You might find it useful to limit the work to some workspace (sandbox in test env for instance) and do not put all objects in a global space:

```
project_name = 'whispyr tests'
workspace = next(ws for ws in whispir.workspaces.list()
                 if ws['projectName'] == 'sandbox')
for contact in workspace.contacts.list():
    print(contact['firstName'])
```

Alternatively, if you know workspace ID:

```
workspace = whispir.workspaces.Workspace(id='C3A1B60DEED39BB3')
```

3.4.2 Message

Message container provides an access to all message's resources such as message statuses and responses:

```
message = workspace.messages.send(to=contact['mri'],
                                  subject='whispyr test',
                                  body='test message, please disregard')
for status in message.statuses.list():
    for category in status['categories']:
        print('{}: {}'.format(category['name'], category['recipientCount']))
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/velimir0xff/whispyr/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

whispyr could always use more documentation, whether as part of the official whispyr docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/velimir0xff/whispyr/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up `whispyr` for local development.

1. Fork the `whispyr` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/whispyr.git
```

3. Install your local copy into a virtualenv. Assuming you have `pipenv` installed, this is how you set up your fork for local development. Run from directory where your fork is cloned to (`whispyr` by default):

```
$ pipenv install --dev
$ pipenv shell
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 whispyr tests
$ py.test
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5, 3.6 and 3.7. Check https://travis-ci.org/velimir0xff/whispyr/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test ./tests/test_whispyr_templates.py
```

Or you can even limit tests to a single test case using a matching expression for `-k` parameter:

```
$ py.test ./tests/test_whispyr_client_basic.py -k test_do_not_retry_qpd
```

4.5 Record VCRs

Whenever you need a new `Whispir` collection (such as `workspaces`) `whispyr` test should be updated accordingly. Collections tests utilises `vcrpy` library. To run tests in recording mode with new (or changed) collections tests should be supplied with all required credentials to be able to talk to whispir.io. You can see them all in a help section of `py.test`. They starts with a `--whispir` prefix:

```
custom options:
--whispir-username=WHISPIR_USERNAME
                      Whispir username
--whispir-password=WHISPIR_PASSWORD
                      Whispir password
--whispir-api-key=WHISPIR_API_KEY
                      Whispir API key
--whispir-gcm-api-key=WHISPIR_GCM_API_KEY
                      Whispir Google Cloud Messaging API key
```

And then configure `py.test` to use credentials to record cassettes:

```
$ py.test ./tests/test_whispyr_devices.py \
    --whispir-username WHISPIR_USERNAME   \
    --whispir-password WHISPIR_PASSWORD   \
    --whispir-api-key WHISPIR_API_KEY     \
    --whispir-gcm-api-key WHISPIR_GCM_API_KEY
```

Once new cassette is recorded please make sure you don't have any sensitive information in them. To automated this process you can install https://github.com/awslabs/git-secrets and add the following patterns into your exclusion list:

```
$ git secrets --add 'apikey=[^&]*'
$ git secrets --add --allowed apikey=V4L1D4P1K3Y
$ git secrets --add --allowed apikey=TEST_API_KEY
```

(continues on next page)

(continued from previous page)

```
$ git secrets --add 'Basic\s+[a-zA-Z0-9=]+'
$ git secrets --add --allowed 'Basic VTUzUk40TTM6UDRaWlcwUkQ='
$ git secrets --add '"gcmApiKey":\s*"[^"]*"'
$ git secrets --add --allowed '"gcmApiKey": "9OO9l3ClouDm355491n94P1K3y"'
```

4.6 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

Credits

5.1 Development Lead

- Grigory Starinkin <starinkin@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.3.0 (2018-08-14)

- Adjust client to new api gateway changes

6.2 0.2.2 (2018-08-06)

- Fix message statuses pagination with detailed view

6.3 0.2.1 (2018-08-03)

- Workaround broken pagination for messagestatus endpoint

6.4 0.2.0 (2018-08-03)

- Add python 2.7 support

6.5 0.1.0 (2018-07-15)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search