pushover_complete Documentation

Release 1.1.1

Scott Colby

Jun 28, 2018

Contents

pushover_complete is a Python package for interacting with *all* aspects of the Pushover API.

To learn more about Pushover and the Pushover API, please visit the Pushover Website, https://pushover.net.

```
>>> from pushover_complete import PushoverAPI
>>> p = PushoverAPI('azGDORePK8gMaC0QOYAMyEEuzJnyUi')  # an instance of the␣
↪PushoverAPI representing your application
>>> p.send_message('uQiRzpo4DXghDmr9QzzfQu27cmVRsG', 'Your toast is finished.')  #␣
↪send a message to a user
```

That's all you need to get started with sending Pushover notifications from your Python program. The majority of Pushover's API endpoints are accessible via *pushover_complete*. Check out the *API Reference* to see how.

On this page:

CHAPTER 1

Installation

1.1 Installation

There are many ways to install a Python package like *pushover_complete*. Here many of those will be explained and the advantages of each will be identified.

If you are not yet familiar with virtual environments, stop reading this documentation and take a few moments to learn. Try some searches for "virtualenv," "virtualenvwrapper," and the "venv" standard library module. I promise that they will change your (Python) life.

1.1.1 Where to Get the Code

From PyPI

Stable releases of *pushover_complete* are located on PyPI, the PYthon Package Index. Installation from here is easy and generally the preferred method:

```
$ pip install pushover_complete
```

From GitHub

`pip` is also able to install from remote repositories. Installation from this project's GitHub repo can get you the most recent release:

```
$ pip install git+https://github.com/scolby33/pushover_complete@master#egg=pushover_
↪complete-latest
```

This works because only release-ready code is pushed to the master branch.

To get the latest and greatest version of *pushover_complete* from the develop branch, install like this instead:

```
$ pip install git+https://github.com/scolby33/pushover_complete@develop#egg=pushover_
↪complete-latestdev
```

In both of these cases, the `#egg=pushover_complete-version` part of the URL is mostly arbitrary. The `version` part is only useful for human readability and the `pushover_complete` part is the project name used internally by `pip`.

From a Local Copy

Finally, `pip` can install from the local filesystem:

```
$ cd /directory/containing/pushover_complete/setup.py
$ pip install .
```

Installing like this lets you make changes to a copy of the project and use that custom version yourself!

1.1.2 Installing in Editable Mode

`pip` has a `--editable` (a.k.a. `-e`) option that can be used to install from GitHub or a local copy in "editable" mode:

```
$ pip install -e .
```

This, in short, installs the package as a symlink to the source files. That lets you edit the files in the `src` folder and have those changes immediately available.

Installation should be as easy as executing this command in your chosen terminal:

```
$ pip install pushover_complete
```

The source code for this project is hosted on Github. Downloading and installing from source goes like this:

```
$ git clone https://github.com/scolby33/pushover_complete
$ cd pushover_complete
$ pip install .
```

If you intend to install in a virtual environment, activate it before running `pip install`.

`pushover_complete` officially supports Python 2.7, 3.5, and 3.6. Currently, Python 3.3 and 3.4 pass all tests and function properly as well, but this could change: these versions are not officially targeted by development. Additionally, due to changes in Sphinx, the documentation cannot be built with Python 3.3.

> **Warning:** Deprecated since version 1.1.0: Support for Python 3.5 is deprecated. It will be removed in the next major version release. This doesn't mean that Python 3.5 will stop working immediately, but I will no longer consider failing tests for that version to be critical.

Support for Python 2.x may be dropped in the future, but only in a major version update (e.g. 1.x.y → 2.x.y) and this change will be announced well in advance.

See *Installation* for further information about installing *`pushover_complete`* in all manner of ways.

CHAPTER 2

Roadmap

2.1 Roadmap

The following Pushover API endpoints are fully implemented:

- /messages.json
- /sounds.json
- /users/validate.json
- /receipts/{receipt}.json
- /receipts/{receipt}/cancel.json
- /subscriptions/migrate.json - /groups/{group_key}.json
- /groups/{group_key}/add_user.json
- /groups/{group_key}/delete_user.json
- /groups/{group_key}/disable_user.json
- /groups/{group_key}/enable_user.json
- /groups/{group_key}/rename.json
- /licenses/assign.json

This constitutes all of the API endpoints available for entities acting as Pushover applications.

New in version 1.1.0: Additionally, the image attachment functionality added to Pushover in January 2018 with version 3.0 of the Pushover apps is now supported.

A command line interface is in the works to allow use directly from your shell.

The Pushover Open Client API may be implemented for a future release.

`pushover_complete` emerged from my frustrating experience with a number of only partially-complete Pushover packages. It is my goal to fully support all of Pushover's API endpoints in this package, beginning with the most essential ones and working from there. The current status of progress towards this goal is tracked in the *roadmap*.

CHAPTER 3

API Reference

Information about each function, class, and method is included here.

3.1 API Reference

This part of the documentation covers all of the interfaces exposed by *pushover_complete*.

3.1.1 The PushoverAPI Class

This class is your gateway to interacting with the Pushover API. Each instance is initialized with a Pushover application token and makes API calls on behalf of that application.

Main Interface

The methods represent most of the useful functions of *PushoverAPI*.

class pushover_complete.**PushoverAPI**(*token*)

The object representing an application interacting with the Pushover API. Instantiated with a Pushover application token. All API calls made via that instance will use the provided application token.

Parameters **token** (*str*) – A Pushover application token

send_message(*user*, *message*, *device=None*, *title=None*, *url=None*, *url_title=None*, *image=None*, *priority=None*, *retry=None*, *expire=None*, *callback_url=None*, *timestamp=None*, *sound=None*, *html=False*)

Send a message via the Pushover API.

Parameters

- **user** (*str*) – A Pushover user token representing the user or group to whom the message will be sent
- **message** (*str*) – The message to be sent

- **device** (*str or list*) – A string or iterable representing the device(s) to which the message will be sent
- **title** (*str*) – The title of the message
- **url** (*str*) – A URL to be included with the message
- **url_title** (*str*) – The link text to be displayed for the URL. If omitted, the URL itself is displayed.
- **image** (*str, pathlib.Path, pathlib2.Path (only in Python 2), or file-like*) – The file path pointing to the image to be attached to the message or a file-like object representing the image data.
- **priority** (*int*) – An integer representing the priority of the message, from -2 (least important) to 2 (emergency). Default is 0.
- **retry** (*int*) – How often the Pushover server will re-send an emergency-priority message in seconds. Required with priority 2 messages.
- **expire** (*int*) – How long an emergency-priority message will be re-sent for in seconds
- **callback_url** (*str*) – A url to be visited by the Pushover servers upon acknowledgement of an emergency-priority message
- **timestamp** (*int*) – A Unix timestamp of the message's date and time to be displayed instead of the time the message is received by the Pushover servers
- **sound** (*str*) – A string representing the sound to be played with the message instead of the user's default. Available sounds can be retreived using *PushoverAPI.get_sounds()*.
- **html** (*int*) – An integer representing if HTML formatting will be enabled for the message text. Set to 1 to enable.

Returns Response body interpreted as JSON

Return type dict

send_messages(*messages*)

Send multiple messages with one call. Utilizes a single HTTP session to decrease overhead.

Parameters **messages** – An iterable of messages to be sent. Each item in the iterable must be expandable using the `**kwargs` syntax with the keys matching the parameters of *PushoverAPI.send_message()*.

Returns Response body interpreted as JSON

Return type list[dict]

get_sounds()

Get the current list of supported sounds from the Pushover servers.

Returns A `dict` of sounds, with keys representing the identifier and values a human-readable name.

Return type dict

validate(*user*, *device=None*)

Validate a user or group token or a user device.

Parameters

- **user** (*str*) – A Pushover user or group token to validate
- **device** (*str*) – A string representing a device name to validate

Returns Response body interpreted as JSON

Return type dict

check_receipt(*receipt*)
Check a receipt issued after sending an emergency-priority message.

Parameters **receipt** (*str*) – The receipt id

Returns Response body interpreted as JSON

Return type dict

cancel_receipt(*receipt*)
Cancel a receipt (and thus further re-sends of the message).

Parameters **receipt** (*str*) – The id of the receipt id to be cancelled

Returns Response body interpreted as JSON

Return type dict

migrate_to_subscription(*user*, *subscription_code*, *device=None*, *sound=None*)
Migrate a user key to a subscription key.

Parameters

- **user** (*str*) – The user key to migrate
- **subscription_code** (*str*) – The subscription code to migrate the user to
- **device** (*str*) – The user's device that the subscription will be limited to
- **sound** (*str*) – The user's preferred sound

Returns Response body interpreted as JSON

Return type dict

migrate_multiple_to_subscription(*users*, *subscription_code*)
Migrate multiple users to subscriptions with one call. Utilizes a single HTTP session to decrease overhead.

Parameters

- **users** – An iterable of messages to be sent. Each item in the iterable must be expandable using the `**kwargs` syntax with keys matching `user` and, optionally, `device` and `sound`. Compare to *PushoverAPI.migrate_to_subscription()*.
- **subscription_code** (*str*) – The subscription code to migrate the user to

Returns Response body interpreted as JSON

Return type dict

group_info(*group_key*)
Retrieve information about a delivery group.

Parameters **group_key** (*str*) – A Pushover group key

Returns Response body interpreted as JSON

Return type dict

group_add_user(*group_key*, *user*, *device=None*, *memo=None*)
Add a user to a group.

Parameters

- **group_key** (*str*) – A Pushover group key

- **user** (*str*) – The user key to be added to the group
- **device** (*str*) – A string representing the device name to add to the group
- **memo** (*str*) – A memo to store with the user's group membership (max 200 characters)

Returns Response body interpreted as JSON

Return type dict

group_delete_user(*group_key*, *user*)
Remove user from a group.

Parameters

- **group_key** (*str*) – A Pushover group key
- **user** (*str*) – The user key to remove from the group

Returns Response body interpreted as JSON

Return type dict

group_disable_user(*group_key*, *user*)
Temporarily disable a user in a group.

Parameters

- **group_key** (*str*) – A Pushover group key
- **user** (*str*) – The user key to disable

Returns Response body interpreted as JSON

Return type dict

group_enable_user(*group_key*, *user*)
Re-enable a user in a group.

Parameters

- **group_key** (*str*) – A Pushover group key
- **user** (*str*) – The user key to enable

Returns Response body interpreted as JSON

Return type dict

group_rename(*group_key*, *new_name*)
Change the name of a group.

Parameters

- **group_key** (*str*) – A Pushover group key
- **new_name** (*str*) – The new name for the group

Returns Response body interpreted as JSON

Return type dict

assign_license(*user_identifier*, *os=None*)
Assign a Pushover license to a user.

Parameters

- **user_identifier** (*str*) – A Pushover user key or email identifying the user to assign the license to

- **os** (*str*) – An OS to limit the license. Available options are `Android`, `iOS`, or `Desktop`

Returns Response body interpreted as JSON

Return type dict

"Private" Methods

These methods, although "private" and used internally by other *PushoverAPI* methods could be useful in some circumstances, particularly when many requests are to be made at one time.

class pushover_complete.**PushoverAPI**

_send_message(*user*, *message*, *device=None*, *title=None*, *url=None*, *url_title=None*, *image=None*, *priority=None*, *retry=None*, *expire=None*, *callback_url=None*, *timestamp=None*, *sound=None*, *html=False*, *session=None*)

The internal function used to send messages via the Pushover API. Takes a `session` parameter to use for sending HTTP requests, allowing the re-use of sessions to decrease overhead. Used to abstract the differences between *PushoverAPI.send_message()* and *PushoverAPI.send_messages()*. Feel free to call directly if your use case isn't fulfilled by the more public methods.

Parameters

- **user** (*str*) – A Pushover user token representing the user or group to whom the message will be sent
- **message** (*str*) – The message to be sent
- **device** (*str or list*) – A string or iterable representing the device(s) to which the message will be sent
- **title** (*str*) – The title of the message
- **url** (*str*) – A URL to be included with the message
- **url_title** (*str*) – The link text to be displayed for the URL. If omitted, the URL itself is displayed.
- **image** (*str, pathlib.Path, pathlib2.Path (only in Python 2), or file-like*) – The file path pointing to the image to be attached to the message or a file-like-object representing the image data.
- **priority** (*int*) – An integer representing the priority of the message, from -2 (least important) to 2 (emergency). Default is 0.
- **retry** (*int*) – How often the Pushover server will re-send an emergency-priority message in seconds. Required with priority 2 messages.
- **expire** (*int*) – How long an emergency-priority message will be re-sent for in seconds
- **callback_url** (*str*) – A url to be visited by the Pushover servers upon acknowledgement of an emergency-priority message
- **timestamp** (*int*) – A Unix timestamp of the message's date and time to be displayed instead of the time the message is received by the Pushover servers
- **sound** (*str*) – A string representing a sound to be played with the message instead of the user's default
- **html** (*int*) – An integer representing if HTML formatting will be enabled for the message text. Set to 1 to enable.

- **session** (*requests.Session*) – A `requests.Session` object to be used to send HTTP requests. Useful to send multiple messages without opening multiple HTTP sessions.

Returns Response body interpreted as JSON

Return type dict

_migrate_to_subscription(*user*, *subscription_code*, *device=None*, *sound=None*, *session=None*)

The internal function to migrate a user key to a subscription key. Takes a `session` parameter to use for sending HTTP requests, allowing the re-use of sessions to decrease overhead. Used to abstract the differences between *PushoverAPI.migrate_to_subscription()* and *PushoverAPI.migrate_multiple_to_subscription()*. Feel free to call directly if your use case isn't fulfilled by the more public methods.

Parameters

- **user** (*str*) – The user key to migrate
- **subscription_code** (*str*) – The subscription code to migrate the user to
- **device** (*str*) – The user's device that the subscription will be limited to
- **sound** (*str*) – The user's preferred sound
- **session** (*requests.Session*) – A `requests.Session` object to be used to send HTTP requests. Useful to send multiple messages without opening multiple HTTP sessions.

Returns Response body interpreted as JSON

Return type dict

_generic_get(*endpoint*, *url_parameter=None*, *payload=None*, *session=None*)

A method for abstracting GET requests to the Pushover API.

Parameters

- **endpoint** (*str*) – The endpoint of the API to hit. Will be joined with "https://api.pushover.net/1/". Example value: "groups/{}.json"
- **url_parameter** (*str*) – A parameter to replace in the endpoint string provided. Example value: "g123456". Combined with the above example value, would result in a final URL of "https://api.pushover.net/1/groups/g123456.json"
- **payload** (*dict*) – A dict of parameters to be appended to the URL, e.g. `{'test-param': False}` would result in the URL having `?test-param=false` appended. Do not include the application token in this dict, as it is added by the function.
- **session** (*requests.Session*) – A `requests.Session` object to be used to send HTTP requests.

Returns Response body interpreted as JSON

Return type dict

_generic_post(*endpoint*, *url_parameter=None*, *payload=None*, *session=None*, *files=None*)

A method for abstracting POST requests to the Pushover API.

Parameters

- **endpoint** (*str*) – The endpoint of the API to hit. Will be joined with "https://api.pushover.net/1/". Example value: "groups/{}.json"

- **url_parameter** (*str*) – A parameter to replace in the endpoint string provided. Example value: "g123456". Combined with the above example value, would result in a final URL of "https://api.pushover.net/1/groups/g123456.json"
- **payload** (*dict*) – A dict of parameters to be appended to the URL, e.g. `{'test-param': False}` would result in the URL having `?test-param=false` appended. Do not include the application token in this dict, as it is added by the function.
- **files** (*dict{str, file-like} or dict{str, tuple(str, file-like[, str[, dict]])}*) – (optional) A dict of `'attachment': value` for attachment to the message. `value` may be a file-like object, or a tuple of at least `('filename', file-like[, 'content_type'[, custom_headers_dict]])`. The optional 'content_type' string describes the file type and custom_headers_dict is a dict-like-object with additional headers describing the file
- **session** (*requests.Session*) – A `requests.Session` object to be used to send HTTP requests.

Returns Response body interpreted as JSON

Return type dict

3.1.2 Exceptions and Errors

exception pushover_complete.**PushoverCompleteError**
Root exception for pushover_complete exceptions. Only used to except any pushover_complete error. Will never be raised explicitly.

exception pushover_complete.**BadAPIRequestError**
An exception raised when Pushover's API responds to a request with an error.

CHAPTER 4

Contributing

4.1 Contributing

There are many ways to contribute to an open-source project, but the two most common are reporting bugs and issue and contributing code.

If you have a bug or issue to report, please visit the issues page on Github and open an issue there.

If you want to make a code contribution, read on for recommendations on how to set up your environment.

Note: Remember to add yourself to `AUTHORS.rst` if you make a code contribution!

4.1.1 Setup

Here's how to get set up to contribute to `pushover_complete`.

1. Fork the `pushover_complete` repository on GitHub (the fork button on the top right!)
2. If your change is small, you may be able to make it directly on GitHub via their online editing process.

   If your change is larger or you want to be able to run tests on your contribution, clone your forked repository locally:

   ```
   $ cd /your/dev/folder
   $ git clone https://github.com/your_username/pushover_complete
   ```

   This will download the contents of your forked repository to `/your/dev/folder/pushover_complete`
3. If you're comfortable with a test-driven style of development, the only thing you need to install is tox, either via the sometimes-temperamental but still useful pipsi (my choice), in a virtual environment, or just system-wide via pip:

```
$ pipsi install tox
# or
$ python -m venv my-virtual-env
$ source my-virtual-env/bin/activate
$ pip install tox
# or
$ pip install tox
```

With `tox` installed, all tests, including checking the `MANIFEST.in` file and code coverage can be performed just by executing:

```
$ tox
```

`tox` handles the installation of all dependencies in virtual environments (under the `.tox` folder) and the running of the tests.

To develop like this, simply write your tests and your code and run `tox` once in a while to check how you're doing.

It is also possible to develop as usual by installing `pushover_complete` in editable mode with pip (preferably in a virtual environment):

```
$ cd /your/dev/folder/pushover_complete
$ cd pip install -e .
```

Tests should still be run via `tox`, but installing the package in this way gives you the flexibility to test things out in the REPL more easily.

4.1.2 Branches

Development of `pushover_complete` follows the "git flow" philosophy of branching. Development takes place on the `develop` branch with individual features being developed on feature branches off of develop. Further reading on this style can be found in this blog post by Jeff Kreeftmeijer. A git plugin to aid in managing branches in this way, called `git-flow`, can be found here.

This might seem a bit complicated, but in general you won't have to worry about it as a contributor. The long and short of this system for you is:

- make a new branch prefixed with "feature/" off of develop before starting work on your contribution (`git checkout -b feature/descriptive-feature-name develop`)
- when pushing changes to your repository, push the right branch! (`git push origin feature/descriptive-feature-name`)

The maintainers will take care of any other issues relating to this.

4.1.3 Pull Requests

Once you've got your feature or bugfix finished (or if it's in a partially complete state but you want to publish it for comment), push it to your fork of the repository and open a pull request against the develop branch on GitHub.

Make a descriptive comment about your pull request, perhaps referencing the issue it is meant to fix (something along the lines of "fixes issue #10" will cause GitHub to automatically link to that issue). The maintainers will review your pull request and perhaps make comments about it, request changes, or may pull it in to the develop branch! If you need to make changes to your pull request, simply push more commits to the feature branch in your fork to GitHub and they will automatically be added to the pull. You do not need to close and reissue your pull request to make changes!

If you spend a while working on your changes, further commits may be made to the main `pushover_complete` repository (called "upstream") before you can make your pull request. In keep your fork up to date with upstream by pulling the changes–if your fork has diverged too much, it becomes difficult to properly merge pull requests without conflicts.

To pull in upstream changes:

```
$ git remote add upstream https://github.com/scolby33/pushover_complete
$ git fetch upstream develop
```

Check the log to make sure the upstream changes don't affect your work too much:

```
$ git log upstream/develop
```

Then merge in the new changes:

```
$ git merge upstream/develop
```

More information about this whole fork-pull-merge process can be found here on Github's website.

4.1.4 Code Style

To make sure your contribution is useful to the overall `pushover_complete` project, you should follow a few conventions.

Run the Tests

Make sure your modifications still pass all tests before submitting a pull requests:

```
$ tox
```

Changes that break the package are mostly useless.

Add New Tests

If you add functionality, you must add tests for it! Untested code is antithetical to reliability. Pull requests that reduce code coverage will likely be rejected. You can check your coverage in the output from `tox`. Lines and files that lack test coverage will be noted there too!

Check out the tests (files that start with `test_` under `src/tests`) to see how previous tests have been written and match your new tests to this style. Tests are performed with `pytest`.

Try and keep your tests simple–tests shouldn't need tests for themselves! Some verbosity in tests isn't the end of the world if it helps to maintain clarity.

Keep Code Changes and Whitespace Cleanup Separate

This is pretty self-explanatory. Code changes and whitespace cleanup should not be mixed–keep them in separate pull requests.

Keep Pull Requests Small

Generally, pull requests should be targeted towards one issue. If you find yourself modifying large swathes of code spanning multiple fixes, thing about splitting your pull request into two (or more!) smaller ones. Large pull requests will likely be rejected.

Follow PEP-8 (ish) and the Zen of Python

If you haven't before, check out the Zen of Python (`python -c 'import this'`) and attempt to keep your code in line with its philosophy. Simple is better than complex!

Keep best practices for formatting Python code in mind when writing your contribution. PEP-8 is generally followed in this project, but not pedantically. Line lengths, for example, are often allowed to creep up if it seems reasonable. If you haven't seen Raymond Hettinger's Beyond PEP 8 presentation, I urge you to go watch it. Unthinking adherence to the "rules" of PEP-8 is not demanded nor is it the best way to write good, Pythonic code.

4.1.5 Making a Release

The steps for making a release of `pushover_complete` are:

1. Create a release branch:

```
$ git flow release start {new_version}
```

2. Bump the version specifier in `src/pushover_complete/__init__.py` and `docs/source/conf.py` from '{new_version}-dev' to plain '{new_version}':

```
$ bumpversion release
```

3. Update the changelog in `docs/source/changelog.rst`, including the last updated date
4. Update the changelog in `README.rst` to match the changelog in the docs
5. Check that any new intersphinx links have corresponding inventory locations in `docs/source/conf.py`. Run

```
$ egrep -rIn --exclude-dir=.eggs --exclude-dir=.tox --exclude-dir=build
↪':\S+:' .
```

   and check for instances of `:meth:`, `:class:`, etc. that are from sources not already included in `intersphinx_mapping` in `conf.py`. (There will be a lot of lines, but with `grep` coloring turned on, it's not that hard to skim through relatively quickly.)

6. Run all tests one last time!

```
$ tox -r
```

   Note: I'm using the `-r` option here, forcing tox to recreate all its virtual environments to be sure this is a "clean" build. It takes longer but I think it's worth it for the peace of mind.

7. Build the project:

```
$ python setup.py sdist bdist_wheel
```

8. Check that the sdist and wheel install properly

> **Warning:** Make sure you do not have any activated virtual environments when running these and the similar test steps. I've gotten inconsistent results in that situation.

```
$ rm -r tmp-virtualenv
$ python -m venv tmp-virtualenv
$ tmp-virtualenv/bin/python -m pip install dist/pushover_complete-{new-version}.
↪tar.gz
$ tmp-virtualenv/bin/python
>>> import pushover_complete
>>> pushover_complete.__version__
'{new_version}'
$ rm -rf tmp-virtualenv
$ python -m venv tmp-virtualenv
$ tmp-virtualenv/bin/python -m pip install dist/pushover_complete-{new-version}-
↪py2.py3-none-any.whl
$ tmp-virtualenv/bin/python
>>> import pushover_complete
>>> pushover_complete.__version__
'{new_version}'
$ rm -rf tmp-virtualenv
```

9. Try a release on the PyPI test server:

```
$ twine upload -r test dist/pushover_complete-{new_version}*
```

Note: This requires a `.pypirc` file in your home folder:

```
[distutils]
index-servers=
    pypi
    test

[test]
repository = https://testpypi.python.org/pypi
username = username
password = password

[pypi]
username = username
password = password
```

Registration with PyPI and TestPyPI is required.

---

10. Test install from the test PyPI:

```
$ rm -rf tmp-virtualenv
$ python -m venv tmp-virtualenv
$ tmp-virtualenv/bin/python -m pip install -i https://testpypi.python.org/pypi␣
↪pushover_complete
$ tmp-virtualenv/bin/python
>>> import pushover_complete
>>> pushover_complete.__version__
```

(continues on next page)

(continued from previous page)

```
'{new_version}'
$ rm -rf tmp-virtualenv
```

11. Check the metadata and such on the test PyPI website
12. Deep breath
13. Upload to PyPI!

```
$ twine upload dist/pushover_complete-{new_version}*
```

14. Test install from PyPI:

```
$ rm -rf tmp-virtualenv
$ python -m venv tmp-virtualenv
$ tmp-virtualenv/bin/python -m pip install pushover_complete
$ tmp-virtualenv/bin/python
>>> import pushover_complete
>>> pushover_complete.__version__
'{new_version}'
$ rm -rf tmp-virtualenv
```

15. Check the metadata and such on the PyPI website
16. Publish the release branch:

```
$ git flow release publish {new_version}
```

17. Finish the release branch:

```
$ git flow release finish {new_version}
```

18. Push the new tag:

```
$ git push --tags
```

19. Attach the sdist and wheel files to the release on GitHub
20. Add changelog notes to the release on GitHub
21. Bump the version to the next dev version:

```
$ bumpversion patch
```

pushover_complete is an open-source project and, so far, is mostly a one-person effort. Any contributions are welcome, be they bug reports, pull requests, or otherwise. Issues are tracked on Github.

Check out *Contributing* for more information on getting involved.

CHAPTER 5

License Information

5.1 License

This software is licensed under the MIT License. The full text of this license is below.

MIT License

Copyright (c) 2018 Scott Colby

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.

`pushover_complete` is licensed under the MIT License, a permissive open-source license.

The full text of the license is available *here* and in the root of the source code repository.

Note: This package is not written by or associated with Superblock, the creators of Pushover. The use of the name "Pushover" in the package name is authorized per Superblock's attribution rules. See the logos section of the Pushover website for more information.

CHAPTER 6

Changelog

6.1 Changelog

pushover_complete adheres to the Semantic Versioning ("Semver") 2.0.0 versioning standard. Details about this versioning scheme can be found on the Semver website. Versions postfixed with '-dev' are currently under development and those without a postfix are stable releases.

Changes as of 6 April 2018

6.1.1 1.1.1 <6 April 2018>

- HOTFIX for 1.1.0
- Fix Python versions badge in the documents index
- Add the Python 3.6 classifier in `setup.py` so the right versions are shown on PyPI

6.1.2 1.1.0 <6 April 2018>

- Add image attachment support (Pulls #5 and #9)
- Officially add support for Python 3.6
- Officially deprecate support for Python 3.5. It will be removed in the next major version release.
- Change default tox environment for Python 3 to py36
- Refactored `.travis.yml` to be more concise and use the new `py` environment specification (Pull #8)
- Some refactoring in the main API (more list comprehensions yay!) (Pull #6)
- Several small documentation changes/refinements

6.1.3 1.0.2 <23 December 2016>

- "Add" Python 3.6 support. It's not in Travis as an allowed failure and didn't require any code changes to pass!
- Fix a major bug with the receipt cancel API. I was using a *GET* request instead of a *POST*
- Stop using the *releases* Sphinx plugin for the changelog. Its philosophy didn't match well with mine
- Update release procedure based on no longer using *releases*
- Some minor documentation fixes

6.1.4 1.0.1 <10 May 2016>

- Officially add Python 2.7 support and add testing for it to tox and Travis
- Numerous updates to documentation and README, etc. to make them prettier and more useful

6.1.5 1.0.0 <9 May 2016>

- Implementation of methods for the Pushover messages, sounds, users, receipt, subscriptions, groups, and licenses APIs
- Documentation and build process

`pushover_complete` adheres to the Semantic Versioning ("Semver") 2.0.0 versioning standard. Details about this versioning scheme can be found on the Semver website. Versions postfixed with '-dev' are currently under development and those without a postfix are stable releases.

You are reading the documents for version 1.1.1 of `pushover_complete`.

Full changelogs can be found on the *Changelog* page.

6.2 TODO

Note: TODO items found in the documentation (marked with the `.. todo::` directive) will be included here automatically.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

G

M

P

S

V