# Contributing to geopy

## Reporting issues

- Please note that the issue tracker is not for questions. Use Stack Overflow

instead. Make sure to tag your question with

[geopy](https://stackoverflow.com/questions/tagged/geopy) tag.

- If possible, before submitting an issue report try to verify that the issue

haven't already been fixed and is not a duplicate.

## Submitting patches

If you contribute code to geopy, you agree to license your code under the MIT.

The new code should follow [PEP8](https://pep8.org/) coding style (except

the line length limit, which is 90) and adhere to the style of

the surrounding code.

You must document any functionality using Sphinx-compatible RST, and

implement tests for any functionality in the `test` directory.

In your Pull Requests there's no need to fill the changelog or AUTHORS,

this is a maintainer's responsibility.

For your convenience the contributing-friendly issues are marked with

`help wanted` label, and the beginner-friendly ones with

`good first issue`.

If your PR remains unreviewed for a while, feel free to bug the maintainer.

### Setup

1. Create a virtualenv

2. Install `geopy` in editable mode along with dev dependencies:

pip install -e ".[dev]"

3. Ensure that tests pass

make test

### Running tests

To run the full test suite:

make test

Or simply:

pytest

To run a specific test module, pass a path as an argument to pytest.

For example:

pytest test/geocoders/nominatim.py

Before pushing your code, make sure that linting passes, otherwise Travis

build would fail:

make lint

### Geocoder credentials

Some geocoders require credentials (like API keys) for testing. They must

remain secret, so if you need to test a code which requires them, you should

obtain your own valid credentials.

Travis CI builds for PRs from forks don't have access to these

credentials to avoid the possibility of leaking the secrets by running

untrusted code. This is a [Travis CI policy][travis\_env]. It means that CI

builds for PRs from forks won't test the code which require credentials.

Such code should be tested locally. But it's acceptable to not test such code

if obtaining the required credentials seems problematic: just leave a note

so the maintainer would be aware that the code hasn't been tested.

[travis\_env]: https://docs.travis-ci.com/user/environment-variables/#Defining-Variables-in-Repository-Settings

You may wonder: why not commit captured data and run mocked tests?

Because there are copyright constraints on the data returned by services.

The credentials can be stored in a json format in a file called `.test\_keys`

located at the root of the project instead of env variables for convenience.

Example contents of `.test\_keys`:

{

"BING\_KEY": "...secret key...",

"OPENCAGE\_KEY": "...secret key..."

}

### Building docs

make docs

Open `docs/\_build/html/index.html` with a browser to see the docs. On macOS you

can use the following command for that:

open docs/\_build/html/index.html

### Adding a new geocoder

Patches for adding new geocoding services are very welcome! It doesn't matter

how popular the target service is or whether its territorial coverage is

global or local.

A checklist for adding a new geocoder:

1. Create a new geocoding class in its own Python module in the

`geopy/geocoders` package. Please look around to make sure that you're

not reimplementing something that's already there! For example, if you're

adding a Nominatim-based service, then your new geocoder class should

probably extend the `geopy.geocoders.Nominatim` class.

2. Follow the instructions in the `geopy/geocoders/\_\_init\_\_.py` module for

adding the required imports.

3. Create a test module in the `test/geocoders` directory. If your geocoder

class requires credentials, make sure to add the `@unittest.skipUnless`

decorator to skip running tests when the required credentials

are not provided (see `test/geocoders/what3words.py` for example).

Refer to the [Geocoder credentials](#geocoder-credentials) section

above for info on how to work with credentials locally.

4. Complete your implementation and tests! Give TDD a try if you aren't used

to it yet! ? Please note that it's possible to run a single test module

instead of running the full suite, which is much slower. Refer to the

[Running tests](#running-tests) section above for a command example.

5. Make sure to document the `geocode` and `reverse` methods with all their

parameters. The class doc should contain a URI to the Terms of Service.

6. Add a reference to that class in the `docs/index.rst` file. Please keep

them ordered alphabetically by their module file names. Build the docs

([Building docs](#building-docs) section above) to make sure that you've

done them right!

That's all!

### Improving a geocoder

If you want to add additional parameters to a `geocode` or `reverse`

method, the additional parameters must be explicitly specified and documented

in the method signature. Validation may be done for type, but values should

probably not be checked against an enumerated list because the service could

change. The additional parameters should go to the end of the method signature.

Please avoid simply passing through arbitrary parameters

(e.g. with `\*\*kwargs`) to the params of a request to the target service.

The target service parameters might change, as well as the service's API,

but the geocoder class's public API should stay the same. It's almost

impossible to achieve that when a pass-through of arbitrary parameters is

allowed.