The Noterator Documentation

Release 0.4.3

James Rutherford

January 20, 2017

Contents

CHAPTER 1

The Noterator

Adding notification to your iteration.

```
>>> from noterator import noterate, EMAIL, TWILIO
>>> for obj in noterate(my_objects, EMAIL|TWILIO):
...     do_something_slow(obj)
...
>>>
```

When the loop completes, The Noterator will notify you by all the methods you passed in. In this case it'll email you and send an SMS to your configured number with Twilio. Other supported notification methods are HipChat (send a notification to a room) and desktop.

You can find more usage information in the usage docs.

1.1 Configuration

Before The Noterator can do anything, you'll need a `config.ini` file (see config.example.ini or the example below to get started).

It's possible to use The Noterator without a configuration file, but it's a little less concise. See the configuration docs for more detail.

By default, we check for `$HOME/.config/noterator/config.ini`, so it's probably best to keep your config there, but you can pass the `config_file` parameter to `noterate` with the path to an alternative location.

You only need to define settings for the methods you wish to use.

```
[desktop]
sound = true

[email]
from_mail = The Noterator <noterator@example.org>
recipient = you@example.org
host = smtp.example.org
port = 25
username = postmaster@example.org
password = password123

[hipchat]
token = abc123
room_id = 123456
from_name = The Noterator
```

```
message_colour = green

[twilio]
account_sid = abc123
token = abc123
from_number = +123456
to_number = +456789
```

1.2 TODO

- New notification plugins: logging, ...
- Notication during iteration, a la tqdm.write

1.3 License

MIT.

1.4 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Keep reading

2.1 Installation

2.1.1 Stable release

To install The Noterator, run this command in your terminal:

```
$ pip install noterator
```

This is the preferred method to install The Noterator, 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.

2.1.2 From sources

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

You can either clone the public repository:

```
$ git clone git://github.com/jimr/noterator
```

Or download the tarball:

```
$ curl -OL https://github.com/jimr/noterator/tarball/master
```

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

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

2.2 Usage

The simplest usage of The Noterator is inside a `for` loop.

```
>>> from noterator import noterate, EMAIL
>>> for obj in noterate(my_objects, EMAIL):
...     do_something_slow(obj)
...
>>>
```

By default, it will notify you by your chosen methods when the iteration completes. You can also provide a description of the iteration that will be included (handy if you're doing several).

```
>>> from noterator import noterate, EMAIL
>>> for obj in noterate(my_objects, EMAIL, "Slow loop 1"):
...     do_something_slow(obj)
...
>>>
```

You can combine notification methods and get notified when iteration begins:

```
>>> from noterator import noterate, EMAIL, TWILIO
>>> for obj in noterate(my_objects, EMAIL|TWILIO, start=True):
...     do_something_slow(obj)
...
>>>
```

If you want to hear about progress before completion, you can use the `every_n` parameter:

```
>>> from noterator import noterate, EMAIL
>>> for obj in noterate(my_objects, EMAIL, every_n=100):
...     do_something_slow(obj)
...
>>>
```

2.2.1 Advanced

If the sequence you're iterating over is an iterator, you can also use `noterate` as one:

```
>>> from noterator import noterate, EMAIL
>>> it = noterate(iter([1,2,3]), EMAIL)
>>> while True:
...     try:
...         result = it.next()
...     except StopIteration:
...         break
...
>>>
```

The `noterate` function is just a wrapper around the `noterator.Noterator` class. If you want to set up a reuesable `Noterator`, you can also do that as follows:

```
>>> from noterator import Noterator, EMAIL
>>> noterator = Noterator(method=EMAIL, every_n=100, start=True)
>>> for obj in noterator(my_objects, desc="loop 1")
...     do_something_slow(obj)
...
>>> for obj in noterator(my_other_objects, desc="loop 2")
...     do_something_else_slow(obj)
...
>>>
```

2.3 Configuration

2.3.1 Available notification methods

Currently, The Noterator supports email, HipChat, Twilio (SMS), and desktop notifications.

Desktop notifications are only supported on Mac & Linux. On Linux, the `notify-send` binary must be on your `$PATH`.

Here is the full set of configuration options available for all notification methods:

```
[desktop]
sound = false

[email]
from_mail = The Noterator <noterator@example.org>
recipient = you@example.org
host = smtp.example.org
port = 25
username = postmaster@example.org
password = password123

[hipchat]
token = abc123
room_id = 123456
from_name = The Noterator
message_colour = green

[twilio]
account_sid = abc123
token = abc123
from_number = +123456
to_number = +456789
```

The only settings that have a default are:

- `email.port`: `25`
- `hipchat.from_name`: `The Noterator`
- `hipchat.message_colour`: `green`
- `desktop.sound`: `false`

2.3.2 Configuration in `.ini` files

By default, `noterator` will look in `$HOME/.config/noterator/config.ini` for configuration. See config.example.ini to get started.

If you want to keep your configuration file somewhere else, you can pass the `config_file` parameter to `noterator`:

```
>>> from noterator import noterate, EMAIL
>>> for obj in noterate(my_objects, method=EMAIL, config_file='/path/to/config.ini'):
...     do_something_slow(obj)
...
>>>
```

2.3.3 Configuration in code

If you set up a `Noterator` class, you can override your file-based configuration per iteration:

```
>>> from noterator import Noterator, EMAIL
>>> noterator = Noterator(method=EMAIL, every_n=100, start=True)
>>> noterator.configure_plugin('email', recipient='someone@example.org')
>>> for obj in noterator(my_objects)
...     do_something_slow(obj)
...
>>> noterator.configure_plugin('email', recipient='someone_else@example.org')
>>> for obj in noterator(my_other_objects)
...     do_something_slow(obj)
...
>>>
```

You can go a step further and avoid using files at all. The following code would work even if no configuration file could be found:

```
>>> from noterator import Noterator, EMAIL
>>> noterator = Noterator(my_objects, method=EMAIL, every_n=100, start=True)
>>> noterator.configure_plugin('email', recipient='you@example.org', from_mail='postmaster@example.or
>>> for obj in noterator:
...     do_something_slow(obj)
...
>>>
```

2.4 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:

2.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/jimr/noterator/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.

Fix Bugs

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

Implement Features

Notification methods in The Noterator are implemented as plugins and we're always open to new methods of notification. If you want to implement a plugin, check the existing ones for what you need to implement. Make sure you update the docs and put example configuration in `config.example.ini`.

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/jimr/noterator/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 :)

2.4.2 Get Started!

Ready to contribute? Here's how to set up *noterator* for local development.

1. Fork the *noterator* repo on GitHub.
2. Clone your fork locally:

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

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv noterator
$ cd noterator/
$ python setup.py develop
```

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 noterator tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

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.

2.4.3 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.

2.4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_noterator
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search