sphinx-no-pragma

Release 0.1

Artur Barseghyan <artur.barseghyan@gmail.com>

Dec 21, 2023

CONTENTS

Improve developer experience:

- Write better docs.
- Do not repeat yourself.
- Assure code low-maintenance.

TL;DR

sphinx-no-pragma is a Sphinx extension for stripping pragma comments from source code used in documentation.

If that's all you need to know to move forward, jump right to the *installation*. Otherwise, read further.

Some say, "documentation is the king". Others argue - "no, demos are". While some say, "testing is everything!" and yet there will be someone else who will jump in with "write clean code! black, isort, mypy and ruff everywhere!"

And yet there's you, who want to be good and write a better package, because there's a generic problem that needs to be solved, and you know how, you want to share it with the world. You also want to assure or at least make an effort in making your project developer friendly, attractive for making contributions, which eventually leads to continuous improvement and make it live long(er).

So, combining the best practices, you:

- Introduce examples in your repository to make it easier to start with.
- Write awesome docs with usage examples (by eventually repeating yourself, copying things from your actual code examples).
- Write tests for your code. Then you realize it's good to test the examples too. Eventually, you have now almost the same code in 3 places: tests, examples and docs.
- Introduce linters and MyPy.

Then you invest your time in making sure all your code looks correct and fix the never-ending MyPy issues.

Then you need to make a small change, which unfortunately, among other, requires altering the examples code. You need to change the examples, the docs, the tests and the examples tests. However, you also need to push the change quickly. As many times before, you skip documentation update, leaving it for "another time".

By that time you discover that code maintenance is a hell. You fix everything, tests pass you're happy to push, by then MyPy starts to nag about issues you have no idea how to solve and by that moment you don't care about them. You're sick of it and start using pragma comments to silence the errors, leaving the fix for another day. You maintenance involves a lot of copy-pasting from one place to another (examples, tests, documentation).

Does this sound familiar?

What if I tell you that actually a couple of steps can be taken out. Namely, that you can use your example code directly in your documentation, using `.. literalinclude::` directive of Sphinx. That part has already been well covered in jsphinx project (JavaScript primarily). However, what jsphinx didn't solve is presence of pragma comments in your

documentation. This project does take care of that part. You don't need to choose or balance between readability, explainability and low-maintenance.

Written by lazy developer for lazy developers to improve developer experience in writing low-maintenance code.

CHAPTER

ONE

FEATURES

- Accurately stips out pragma comments from your source code that you include in your documentation.

CHAPTER

TWO

PREREQUISITES

Python 3.8+

CHAPTER

THREE

INSTALLATION

```
pip install sphinx-no-pragma
```

CHAPTER

FOUR

DOCUMENTATION

- Documentation is available on Read the Docs.
- For guidelines on contributing check the Contributor guidelines.

CHAPTER

FIVE

USAGE EXAMPLE

In order to move forward, you first need to get educate yourself a little on Sphinx's directives. Namely the `.. literalinclude::` and `:download:`. For that, first read the jsphinx documentation.

But there might be a little problem with that. Of course you might be lucky and have zero pragma comments in your code (no `# noqa`, no `# type: ignore`, etc). But more often, you get at least a couple of these. Your perfectionist nature doesn't easily allow you to let them be part of your concise, beautiful documentation. Cursing me for earlier advices, you start to replace your DRY documentation part with copy-pasted examples.

This is where this package jumps in. It simply is a Sphinx extension that strips all pragma comments from your code that goes into documentation.

5.1 Sphinx configuration

docs/conf.py

```
extensions = [
    # ... other extensions
    "sphinx_no_pragma",
    # ... other extensions
]
```

5.2 Code example

examples/example_1.py

```
from typing import Any, Optional

class ThirdPartyLibrary:
    @staticmethod
    def get_dynamic_object() -> Any:
        # Returns an object whose type is not known at compile time
        return "a string"  # In reality, this could be any type


# Usage of the third-party library
obj = ThirdPartyLibrary.get_dynamic_object()

# Attempt to use the object as a string, even though its type is 'Any'
```

(continues on next page)

(continued from previous page)

```
length = len(obj)  # type: ignore

# Deliberately long line to violate PEP 8 line length rule, suppressed with noqa
print(f"The length of the object, a dynamically typed one, is just {length}")  # noqa
```

Given that this is your code structure:

```
├── examples
│   └── example_1.py
├── docs
│   ├── conf.py
│   ├── index.rst
│   ├── Makefile
│   ├── _static
│   │   └── example_1.py
│   └── usage.rst
├── LICENSE
├── Makefile
├── pyproject.toml
├── README.rst
└── sphinx_no_pragma.py
```

Either use `html_extra_path = ["examples"]` or make a symlink to `examples/example_1.py` from `docs/_static`.

Then include it in your docs as follows:

```
.. container:: jsphinx-download

.. literalinclude:: _static/example_1.py
    :language: python
    :lines: 1-

*See the full example*
:download:`here <_static/example_1.py>`
```

Now, rendered, your code will not contain *# type: ignore* or *# noqa* pragma comments.

See the demo. Click on the *See the full example here* link to see the original code.

CHAPTER

SIX

TESTS

Run the tests with unittest:

```
python -m unittest sphinx_no_pragma.py
```

Or pytest:

```
pytest
```

CHAPTER

SEVEN

LICENSE

MIT

CHAPTER

EIGHT

SUPPORT

For security issues contact me at the e-mail given in the *Author* section.

For overall issues, go to GitHub.

CHAPTER

NINE

AUTHOR

Artur Barseghyan <artur.barseghyan@gmail.com>

CHAPTER

TEN

PROJECT DOCUMENTATION

Contents:

Table of Contents

10.1 Demo

example_1.py

```
from typing import Any, Optional


def my_func(arg1: Optional[Any] = None) -> int:
    # This is a very long line that should normally fail, but we want it to
    # be present as is.
    my_very_very_very_long_variable_name_just_to_show_a_very_long_line_of_x_characters =␣
↪(
        1
```

(continues on next page)

(continued from previous page)

```
    )
    print(
        my_very_very_very_long_variable_name_just_to_show_a_very_long_line_of_x_
↪characters
    )
    a = (
        arg1
        or my_very_very_very_long_variable_name_just_to_show_a_very_long_line_of_x_
↪characters
    )
    print(a)

    return "0"


class ThirdPartyLibrary:
    @staticmethod
    def get_dynamic_object() -> Any:
        # Returns an object whose type is not known at compile time
        return "a string"  # In reality, this could be any type


# Usage of the third-party library
obj = ThirdPartyLibrary.get_dynamic_object()

# Attempt to use the object as a string, even though its type is 'Any'
length = len(obj)

# Deliberately long line to violate PEP 8 line length rule, suppressed with
print(
    f"The length of the object, a dynamically typed one, is just {length}"
)
```

See the full example here

10.2 Security Policy

10.2.1 Reporting a Vulnerability

Do not report security issues on GitHub!

Please report security issues by emailing Artur Barseghyan <artur.barseghyan@gmail.com>.

10.2.2 Supported Versions

Make sure to use the latest version.

The two most recent `sphinx-no-pragma` release series receive security support.

For example, during the development cycle leading to the release of `sphinx-no-pragma` 0.17.x, support will be provided for `sphinx-no-pragma` 0.16.x.

Upon the release of `sphinx-no-pragma` 0.18.x, security support for `sphinx-no-pragma` 0.16.x will end.

Version	Supported
0.1.x	Yes
< 0.1	No

10.3 Contributor guidelines

10.3.1 Developer prerequisites

pre-commit

Refer to pre-commit for installation instructions.

TL;DR:

```
pip install pipx --user  # Install pipx
pipx install pre-commit  # Install pre-commit
pre-commit install  # Install pre-commit hooks
```

Installing pre-commit will ensure you adhere to the project code quality standards.

10.3.2 Code standards

black, isort, ruff and doc8 will be automatically triggered by pre-commit. Still, if you want to run checks manually:

```
make black
make doc8
make isort
make ruff
```

10.3.3 Requirements

Requirements are compiled using pip-tools.

```
make compile-requirements
```

10.3.4 Virtual environment

You are advised to work in virtual environment.

TL;DR:

```
python -m venv env
pip install -e .[all]
```

10.3.5 Documentation

Check documentation.

10.3.6 Testing

Check testing.

If you introduce changes or fixes, make sure to test them locally using all supported environments. For that use tox.

```
tox
```

In any case, GitHub Actions will catch potential errors, but using tox speeds things up.

10.3.7 Pull requests

You can contribute to the project by making a pull request.

For example:

- To fix documentation typos.
- To improve documentation (for instance, to add new rule or fix an existing rule that doesn't seem to work).
- To introduce a new feature.

General list to go through:

- Does your change require documentation update?
- Does your change require update to tests?

When fixing bugs (in addition to the general list):

- Make sure to add regression tests.

When adding a new feature (in addition to the general list):

- Make sure to update the documentation (check whether the installation, features or demo require changes).

10.3.8 Questions

Questions can be asked on GitHub discussions.

10.3.9 Issues

For reporting a bug or filing a feature request use GitHub issues.

Do not report security issues on GitHub. Check the support section.

10.4 Contributor Covenant Code of Conduct

10.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

10.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

10.4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

10.4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

10.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at artur.barseghyan@gmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

10.4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

10.4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

10.5 Release history and notes

Sequence based identifiers are used for versioning (schema follows below):

```
major.minor[.revision]
```

- It's always safe to upgrade within the same minor version (for example, from 0.3 to 0.3.4).
- Minor version changes might be backwards incompatible. Read the release notes carefully before upgrading (for example, when upgrading from 0.3.4 to 0.4).
- All backwards incompatible changes are mentioned in this document.

10.5.1 0.1

2023-12-18

- Initial beta release.

10.6 Package

10.6.1 sphinx_no_pragma module

https://github.com/barseghyanartur/sphinx-no-pragma/

class sphinx_no_pragma.**NoPragmaLiteralInclude**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Bases: LiteralInclude

remove_endings(*line*, *endings*)

run()

sphinx_no_pragma.**setup**(*app*)

10.7 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX