# Setting Up Sphinx Documentation

# Using GitHub Only (No Command Line)

for`JsonGrapher-py`

I’m documenting every step of how I set up Sphinx documentation for `JsonGrapher-py`, using only the GitHub web interface because it has everything that i need, all steps cannot be completed using just the desktop app

Below, there is some copying and pasting into new files. This was the Main Source for the copy and pasting into files, such as yaml-

https://medium.com/@ariassbustojonathan/automate-python-documentation-with-sphinx-for-github-3dda2ca690ec

---

**Step 1: Starting with a New Git Branch**

Before making any documentation-related changes, I want to create a separate branch. This keeps the main branch clean and lets me experiment without risk.

Firstly

1. I opened my GitHub repository in the browser.

2. At the top-left, next to the branch dropdown (usually says `main`), I clicked it and selected “Create branch”

3. I named the new branch:

sphinx-doc-setup

4. GitHub automatically switched me to the new branch. Now, any file I add or edit will be isolated here.

---

**Step 2: Understanding and Planning Folder Structure**

Before creating files, I needed to understand how Sphinx expects things to be organized. I didn’t want to rely on partial information, so I asked ChatGPT:

> “tell me about only the folder structure that the user has to make while creating a sphinx documentation”

ChatGPT responded with this layout:

project\_root/

├── docs/

│ ├── source/

│ │ ├── conf.py

│ │ ├── index.rst

│ │ ├── \_static/

│ │ ├── \_templates/

│ │ └── other\_docs.rst

│ └── build/

├── your\_code/

└── README.md

That gave me a good idea of what’s expected. I don’t need `\_static/`, `\_templates/`, or `other\_docs.rst` right now, so I will skip those. I’ll start by creating only `docs/conf.py`.

---

**Step 3: Creating `conf.py`**

Next, I need to create the main configuration file Sphinx uses — `conf.py`.

To figure out what to include inside `conf.py`, I used the example at:

[https://sphinxcontrib-jupyter.readthedocs.io/en/latest/config-example.html

That example was meant for more complex setups (like Jupyter notebooks), so I adapted it to keep only what’s needed for basic documentation.

---

Action:

1. In GitHub, I clicked \*\*“Add file” > “Create new file”\*\*.

2. In the filename box, I typed:

docs/source/conf.py

This automatically creates the `docs/` folder and places the `conf.py` file inside it.

---

What I’m Adding to `conf.py` and Why

Here’s the content I’m putting inside the file, along with explanations:

Because I want Sphinx to understand \*\*Google-style docstrings\*\*, I need to enable the \*\*Napoleon\*\* extension. According to its documentation:

[<https://sphinxcontrib-napoleon.readthedocs.io/en/latest/>]

> "Napoleon is a Sphinx extension that enables Sphinx to parse both NumPy and Google style docstrings."

Also, I need a theme. I chose the popular \*\*Read the Docs theme\*\*, which looks good out of the box and is used widely. I found the theme’s GitHub page here:

[<https://github.com/readthedocs/sphinx_rtd_theme>]

I’m using the theme by specifying:

html\_theme = 'sphinx\_rtd\_theme'

Other themes are available, but this one is clean and responsive, so I’ll start with it.

Action:

Code I Pasted into docs/source/conf.py:

import os

import sys

# Add the parent directory to sys.path so Sphinx can find the source code

sys.path.insert(0, os.path.abspath('../../'))

# Project information

project = 'JsonGrapher'

copyright = '2025, JsonGrapher'

author = 'JsonGrapher'

release = '0.1'

# Extensions to enable

extensions = [

"sphinx.ext.napoleon",

"sphinx.ext.autodoc"

]

# HTML theme to use

html\_theme = 'sphinx\_rtd\_theme'

html\_static\_path = ['\_static']

After pasting the code:

I clicked Commit new file.

That completed the creation of conf.py.

Summary of Sources Used During This Step

Folder structure provided by ChatGPT

config structure adapted from

https://sphinxcontrib-jupyter.readthedocs.io/en/latest/config-example.html

(kept only essential lines)

Napoleon extension documentation:

<https://sphinxcontrib-napoleon.readthedocs.io/en/latest/>

Theme source:

https://github.com/readthedocs/sphinx\_rtd\_theme

**Step 4: Creating the `index.rst` Homepage**

Now that I’ve created the `conf.py` configuration file inside the `docs/` folder, the next step is to create the documentation homepage — `index.rst`.

This file will serve as the landing page for the entire documentation site. It also acts as the root of the table of contents (sidebar navigation) for all other documentation pages.

To figure out what this file should contain and how it works, I referred to the official Sphinx documentation here:

> https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html

From that, I learned:

- `index.rst` is treated as the main entry point by Sphinx.

- It can include text, links, and a special directive called `.. toctree::` to define the navigation menu.

- Even if other files like `install.rst` or `modules.rst` don’t exist yet, I can reference them here and add them later.

---

Action:

1. On GitHub (still on the `sphinx-doc-setup` branch), I clicked \*\*“Add file” > “Create new file”\*\*.

2. In the filename input, I typed:

docs/source/index.rst

This places the file right alongside `conf.py`, keeping everything organized in the `docs/` directory

3. In the main content editor, I pasted the following content into index.rst:

---

.. JSONGrapher documentation master file

Welcome to JSONGrapher’s Documentation

======================================

JSONGrapher is a Python library for visualizing and working with JSON data

through graph-based structures.

This documentation provides a complete reference to its modules and usage.

.. contents:: Table of Contents

:local:

:depth: 2

API Reference

=============

.. toctree::

:maxdepth: 3

:caption: Modules

:glob:

modules

JSONGrapher/\*

Why I Wrote It This Way

The main heading (JSONGrapher Documentation) becomes the visible title of the page.

The welcome message helps users understand the purpose of the package.

The three bulleted links act as shortcuts to:

The GitHub repo

An installation guide (will be written in install.rst)

The API reference (will be generated or written in modules.rst)

The .. toctree:: block is critical:

It defines the documentation’s navigation menu.

The :maxdepth: 2 setting allows up to two levels of nesting in the sidebar.

The :caption: appears as a heading above the linked pages in the sidebar.

install and modules are .rst files I’ll create later — for now, they act as placeholders.

Even though those two files (install.rst, modules.rst) don’t exist yet, including them early helps define the structure of the documentation in advance.

Then I clicked Commit new file.

The documentation homepage (docs/index.rst) is now created and saved.

Sources Used in This Step

Official Sphinx guide to index.rst and toctree:

<https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html>

sublime-and-sphinx-guide:

https://sublime-and-sphinx-guide.readthedocs.io/en/latest/indices.html

**Step 5: Setting Up GitHub Actions to Automatically Build and Deploy the Docs**

Now that the basic documentation files (`conf.py`, `index.rst`) are in place, the next step is to \*\*automate the documentation build process\*\* using GitHub Actions.

This way, every time I push changes to the repo, the documentation will:

1. Be generated from Python source code docstrings

2. Be built using Sphinx

3. Be published to GitHub Pages

---

Where I Got This Setup From

To do this, I needed a GitHub Actions workflow. I combined two sources:

- \*\*Sphinx GitHub Action on the Marketplace\*\*:

https://github.com/marketplace/actions/sphinx-docs

→ This gave me the base action structure.

- \*\*Amar Askar’s `sphinx-action` repo\*\*:

https://github.com/ammaraskar/sphinx-action

→ This helped fill in the rest, including how to set up deployment and build the docs.

---

Action: Creating the Workflow File

1. I went to the GitHub repository (still on my `sphinx-doc-setup` branch).

2. I clicked \*\*“Add file” > “Create new file”\*\*.

3. In the file name field, I typed the following \*\*exactly\*\*:

.github/workflows/documentation.yml

- This will create a `.github/` folder (if it doesn't exist)

- Inside it, a `workflows/` folder

- And finally the workflow file: `documentation.yml`

---

Contents of `.github/workflows/documentation.yml`

This file defines the full GitHub Actions pipeline for building and deploying the documentation.

Here is the code I used, it can be copy and pasted:

name: documentation

on: [push, pull\_request, workflow\_dispatch]

permissions:

contents: write

jobs:

docs:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: actions/setup-python@v5

- name: Install dependencies

run: |

pip install sphinx sphinx\_rtd\_theme

- name: Generate .rst files from source code

run: |

cd docs/source

sphinx-apidoc -o . ../../JSONGrapher -f

- name: Build Sphinx docs

run: |

cd docs/source

python -m sphinx -b html . ../build/html

- name: Deploy to GitHub Pages

uses: peaceiris/actions-gh-pages@v3

if: ${{ github.event\_name == 'push' && github.ref == 'refs/heads/sphinx-doc-setup' }}

with:

publish\_branch: gh-pages

github\_token: ${{ secrets.GITHUB\_TOKEN }}

publish\_dir: docs/build/html

force\_orphan: true

############

Explanation of Each Step

on: [push, pull\_request, workflow\_dispatch]

→ This runs the workflow when code is pushed, a PR is created, or manually triggered.

sphinx-apidoc

→ This is the tool that automatically generates .rst files from Python docstrings. It scans the code and produces .rst files for each module.

python -m sphinx -b html

→ This builds the HTML documentation from .rst files.

peaceiris/actions-gh-pages@v3

→ This action deploys the built HTML files to a special gh-pages branch, which is what GitHub Pages will use to serve the docs.

publish\_dir: docs/build/html

→ That’s where Sphinx outputs the generated website — we specify that as the folder to publish.

if: ${{ github.event\_name == 'push' && github.ref == 'refs/heads/sphinx-doc-setup }}

→ This means the deploy step only happens when I push to the sphinx-doc-setup branch. You must change "sphinx-doc-setup" to the real name of the branch you’re working in

Important Note: \_\_init\_\_.py Required

Before this will work, the folder containing your actual Python code (e.g. JSONGrapher/) must have an \_\_init\_\_.py file inside it.

Without this file:

Python won’t recognize your source code as a package

sphinx-apidoc won’t treat the .py files as importable modules

The generated documentation will be empty or broken

Even an empty \_\_init\_\_.py is fine.

I Committed the File.

Once the code above was pasted into GitHub:

Now, every time I push changes to the documentation or code, GitHub will automatically:

Extract the docstrings

Generate the docs

Build the site

Deploy to GitHub Pages (when pushed to the correct branch)

Sources Used in This Step

Sphinx GitHub Action (Marketplace version):

https://github.com/marketplace/actions/sphinx-docs

Amar Askar’s advanced Sphinx GitHub Action template:

https://github.com/ammaraskar/sphinx-action

Deployment action (peaceiris GH Pages):

<https://github.com/peaceiris/actions-gh-pages>

**Step 6: Enabling GitHub Pages and Making the Docs Public**

After setting up the GitHub Actions workflow to build and deploy the docs (in the previous step), the next task is to \*\*make the documentation site publicly visible\*\* via GitHub Pages.

This part involves:

- Triggering the workflow at least once

- Allowing it to create the `gh-pages` branch

- Configuring GitHub Pages to serve from that branch

---

**Step 7.1 – Triggering the Workflow for the First Time**

Before GitHub can deploy the site, the workflow must run at least once and successfully generate the documentation. To force this first run, I created a dummy file in the `docs/` folder.

Action:

1. In the GitHub repo (still on the `sphinx-doc-setup` branch), I clicked \*\*“Add file” > “Create new file”\*\*.

2. In the filename box, I typed:

docs/.trigger

The leading dot makes it a hidden file — it won’t interfere with the documentation build process.

3. In the file content box, I added a single comment

4. THen Clicked \*\*Commit new file\*\*.

The workflow was now triggered. I confirmed this by going to the \*\*Actions\*\* tab and watching the build start.

---

**### Step 7.2 – What to Do When You Make Changes**

After this initial trigger, \*\*every time I update any `.py`, `.yml`, `.rst`, or docstring\*\*, I need to re-trigger the workflow.

GitHub Actions \*\*won’t run the Sphinx deploy workflow just because `.rst` files or docstrings changed\*\*. It only reruns if there’s a real commit — so I need a reliable way to force it.

#### Solution:

Every time I make a documentation-related change (like fixing a docstring or updating an `.rst` page), I do the following:

1. Open `docs/.trigger`

2. Make a minimal edit — for example:

- Remove a space or letter, then type it back.

- Change the comment to have new ate, like from:

```

# This is the first trigger file commit, Aug 13 2025

```

to:

```

# This is the trigger file commit on Aug 28th, 2025.

```

3. Commit that change with:

Trigger rebuild of documentation after source edits

This creates a new commit, which re-runs the GitHub Action and regenerates the documentation site.

---

**### Step 7.3 – Setting Up GitHub Pages**

Once the workflow runs successfully the first time, a new branch called `gh-pages` will be automatically created by the GitHub Action. This branch contains the built HTML files.

Now I needed to tell GitHub to serve those files as a website.

Action:

1. I went to the \*\*Settings\*\* tab of the repository.

2. Scrolled down to the \*\*“Pages”\*\* section (in the left-hand menu).

3. Under \*\*“Source”\*\*, I selected:

- \*\*Branch:\*\* `gh-pages`

- \*\*Folder:\*\* `/ (root)`

4. Saved the changes.

Once done, GitHub displayed the live documentation URL near the top of the “Pages” section of Settings.

Example:

https://AdityaSavara.github.io/jsongrapher-py/

https://yourusername.github.io/your-repo-name/

Now the documentation site is online and auto-updates when I push new commits and trigger the workflow again.

---

Sources Used in This Step

- \*\*GitHub Pages documentation\*\*:

https://docs.github.com/en/pages/getting-started-with-github-pages

- \*\*GitHub Actions + Pages deployment guides\*\*:

https://github.com/peaceiris/actions-gh-pages

https://github.com/ammaraskar/sphinx-action

**Step 8: Debugging Problems During Setup**

After setting up the folder structure, the configuration file, the `index.rst`, and the GitHub Actions workflow, I started running into some issues. This section documents the problems I encountered while setting everything up — and how I solved them — so that if the same issues come up in the future, I’ll know what to do.

---

Problem 1: `make html` Command Failed in the Workflow

\*\*What Happened:\*\*

When the GitHub Actions workflow ran, it failed on the step that was supposed to build the documentation. I had originally used the standard command:

make html

Why It Failed:

This command requires a Makefile to be present in the docs/ folder, which I didn't create (and didn’t want to, since I was avoiding command-line tools).

How I Fixed It:

Instead of using make, I rewrote the step to explicitly use Python:

python -m sphinx -b html source build/html

This directly tells Python to run Sphinx and build the HTML version of the docs, specifying source/ as the input and build/html/ as the output folder. Once I committed this change to the .yml file, the build step succeeded.

Problem 2: Docstrings Not Showing in the Generated Docs

What Happened:

Even after the build succeeded, the output HTML did not include any of my Python function/class docstrings — only the index.rst content was visible.

How I Debugged It:

I went to the Actions tab on GitHub and opened the logs for the most recent run. I looked at the output from the sphinx-apidoc and sphinx-build commands.

In the logs, I noticed an error related to bad indentation in one of the generated .rst files — likely caused by docstrings in the Python code being indented inconsistently.

How I Fixed It:

I opened the affected Python file and manually fixed the indentation inside the docstring. Specifically, I made sure that all docstring sections (like parameters and returns) were consistently indented

After committing that change and triggering a rebuild, the documentation displayed all the docstrings correctly.

Problem 3: Merge Conflict During Branch Workflow

What Happened:

While trying to merge the sphinx-doc-setup branch into main, GitHub showed a merge conflict.

Why It Happened:

The .github/workflows/documentation.yml file had been edited in both branches — and GitHub couldn’t automatically decide which version to keep.

How I Fixed It:

I followed GitHub’s merge conflict UI instructions, manually resolved the changes (kept the version with my workflow edits), and committed the result. The conflict was cleared without issue.

Note: Behavior of Napoleon Extension with Args vs Parameters

What I Noticed:

Sphinx’s Napoleon extension automatically converts Google-style Args: into “Parameters” in the output HTML.

Example:

If I write this in a docstring:

"""

Args:

name (str): The name of the user.

age (int): The age of the user.

"""

Napoleon renders it under the heading Parameters, not Args.

Why That Happens:

This is the intended behavior of Napoleon, as documented here:

https://sphinxcontrib-napoleon.readthedocs.io/en/latest/

It normalizes docstring formats (Google and NumPy) into reStructuredText, and uses the word "Parameters" consistently.

What I Decided:

I chose to keep it that way. It’s recommended and keeps the documentation consistent with most Python projects.

Alternative (Not Used):

If I ever wanted to show “Args” instead of “Parameters”, I would have to:

Turn off Napoleon’s automatic conversion

Or manually rewrite all my docstrings in raw .rst style (which is not ideal)

Sources and Tools Used in This Step

GitHub Actions Logs (real-time log viewer) — used to debug build failures and inspect errors

Napoleon Extension Docs (handling Google/NumPy docstrings):

https://sphinxcontrib-napoleon.readthedocs.io/en/latest/

### **9. Changing the trigger to work on main branch only**

In order to do this you just have to change the documentation.yml file a bit so that

> Push events only run when the branch is main.  
 > You avoid unnecessary runs from other branches entirely.

Here is the full .yml code tailored specifically for the main branch:

name: documentation

on:

push:

branches:

- main # only run on push to main

workflow\_dispatch: # still allow manual trigger

permissions:

contents: write

jobs:

docs:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: actions/setup-python@v5

- name: Install dependencies

run: |

pip install sphinx sphinx\_rtd\_theme

- name: Generate .rst files from source code

run: |

cd docs/source

sphinx-apidoc -o . ../../JSONGrapher -f

- name: Build Sphinx docs

run: |

cd docs/source

python -m sphinx -b html . ../build/html

- name: Deploy to GitHub Pages

uses: peaceiris/actions-gh-pages@v3

if: ${{ github.ref == 'refs/heads/main' }} # deploy only when on main

with:

publish\_branch: gh-pages

github\_token: ${{ secrets.GITHUB\_TOKEN }}

publish\_dir: docs/build/html

force\_orphan: true

### 

### **My Recommendation for Branching and Merging Sphinx Documentation:**

1. **Normal process for working on main fork**
   1. Make a new branch (or work on existing branch).
   2. Add function, including docstrings.
   3. Test
   4. Pull and merge into main branch.
2. **Normal process for working on other-fork**
   1. Make a new branch (or work on existing branch)
   2. Add function, including docstrings
   3. Test
   4. Make a new branch on main fork
   5. Merge from other-fork branch to new branch on main fork
   6. Test
   7. Merge into main branch of main fork

**#3 and #4 is how to make the workflow process after sphinx setup.**

1. **With Sphinx ‘installed’ process for working on main fork**
   1. Make a new branch (or work on existing branch).
   2. Add function, including docstrings.
   3. Test
   4. Pull and merge into main branch.
2. **With Sphinx ‘installed process for working on other-fork**
   1. Make a new branch (or work on existing branch)
   2. Add function, including docstrings
   3. Test
   4. Make a new branch on main fork
   5. Merge from other-fork branch to new branch on main fork
   6. Test
   7. Merge into main branch of main fork

## **Triggering Sphinx Builds (Best Practice)**

* **Trigger the workflow at the end of each coding session**, especially after adding functions, fixing bugs, or updating docstrings.
* This ensures the **live documentation site stays up-to-date** with your latest changes.
* The website only updates when the Sphinx workflow is triggered — **committing alone is not enough**.

### **Fastest Way to Trigger**

* Edit the .trigger file **directly on GitHub**.
* Just **add or remove a character**, commit, and it will run the workflow.
* This avoids unnecessary commits to real files.