Modern, interactive and impactful software documentation with Sphinx, MyST and the Diátaxis framework

Ghislain Vaillant

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Inria

Modern, interactive and impactful software documentation

with Sphinx, MyST and the Diátaxis framework

Ghislain Vaillant

Target audience

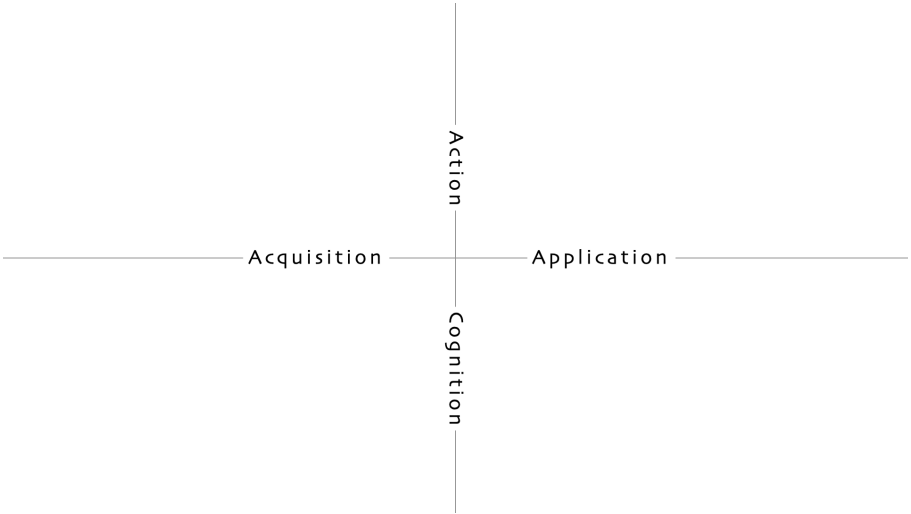
How to write **good** software documentation?

Sommaire

1. Impactful documentation with Diátaxis
2. Modern documentation with Sphinx and MyST
3. Tooling and automation

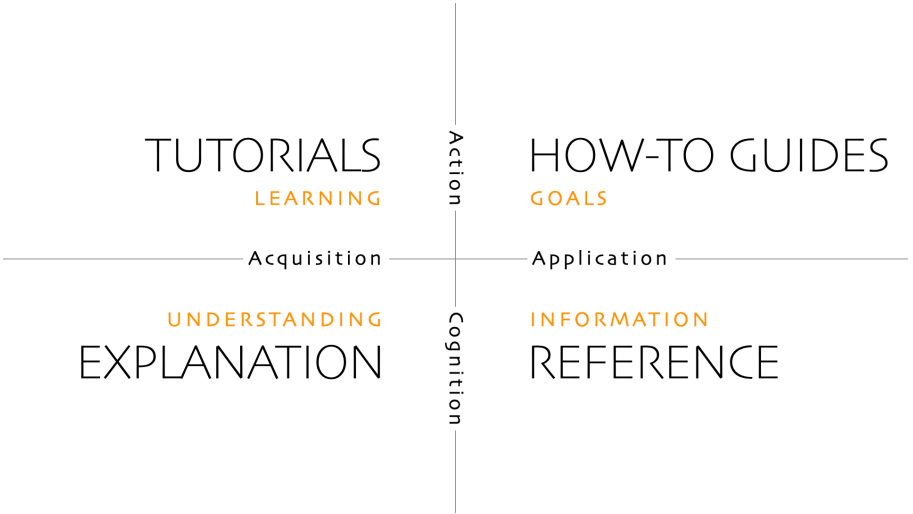
01
Impactful documentation with Diátaxis

Domain of craft



[1] Procida, D. Diátaxis documentation framework. https://diataxis.fr/

Map of documentation



[1] Procida, D. Diátaxis documentation framework. https://diataxis.fr/

Cycle of needs

[1] Procida, D. Diátaxis documentation framework. https://diataxis.fr/

Overlapping needs

Practitioner's needs	addressed by	and by
Develop practical knowledge	Tutorials	Cookbook
Serve the application of skills	Reference	Cookbook
Provide theoretical knowledge	Reference	Explanation
Serve the acquisition of skills	Tutorials	Explanation

Distinctive concerns

Material	Guidance	Expectations
Tutorials	Lesson	Teach, lead, educate
Cookbook	Goals	Guide, follow, achieve
Reference	Information	State, inform, describe
Explanation	Topic	Understand, discuss, clarify

02
Modern documentation with Sphinx and MyST

Getting started

1. Install the base set of dependencies

   ```
   pip install sphinx myst-parser
   ```

2. Bootstrap the documentation project

   ```
   sphinx-quickstart docs
   ```

3. Enable support for MyST

   ```
   # docs/conf.py
   extensions = ["myst_parser"]
   ```

4. Start editing `docs/index.md`

MyST Markdown

MyST Markdown = CommonMark + MyST *roles* and *directives*

Example of a MyST role

````
For more information, please check this section
{ref}`some-section`.
````

Example of a MyST directive

````
```{admonition} Warning title
:class: warning

Some details about the warning
```
````

MyST extensions

Optional MyST syntax enhancements can be enabled through extensions[1].

Enable code fences using colons

```
# docs/conf.py
myst_enable_extensions = ["colon_fence"]
```

Admonition rendered as Markdown

```
:::{admonition} Warning title
:class: warning

Some **details** about the _warning_
:::
```

[1] https://myst-parser.readthedocs.io/en/latest/syntax/optional.html

Sphinx extensions

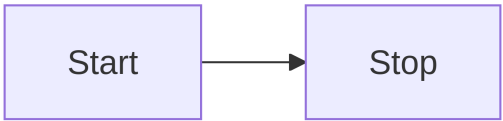
Sphinx extensions are usable with MyST markdown.

Enable support for mermaid graphs

```
# docs/conf.py
extensions = ["sphinxcontrib.mermaid"]
```

Using the mermaid directive

````
```{mermaid}
flowchart LR
    Start --> Stop
```
````



Autogenerated reference documentation

Use the `sphinx-autoapi`[2] extension to automatically generate reference documentation from docstrings.

Reference documentation will be composed from module / class / function docstrings and cached for faster builds. Unlike `sphinx-autodoc`, it does not rely on package imports.

Configured with

```
# docs/conf.py
extensions = ["autoapi.extension"]

autoapi_dirs = ["../src"]
autoapi_root = "reference"
```

[2] https://sphinx-autoapi.readthedocs.io/

Literate programming with MyST notebooks

Use the `myst-nb`[3] extension to support writing and versioning Jupyter notebooks in MyST Markdown.

Configured with

```
# docs/conf.py
extension = ["myst_nb"]
```

Executable code block

````
```{code-block}
print("hello world!")
```
````

[3] https://myst-nb.readthedocs.io

Interactive notebooks with sphinx-thebe

Use the `sphinx-thebe`[4] extension to support interactive execution of notebooks with Binder[5].

Configured with

```
# docs/conf.py
extension = [
        "myst-nb",
        "sphinx-thebe",
]
```

Launch button

````
```{thebe-button}
```
````

Interactive code block

````
```{code-block}
:class: thebe

print("hello world!")
```
````

[4] https://sphinx-thebe.readthedocs.io
[5] https://mybinder.org/

03
Tooling and automation

Project setup

- Bootstrap Sphinx project with sphinx-quickstart
- Setup interactive editing with sphinx-autobuild
- Use a separate group for documentation dependencies

```toml
# pyproject.toml
[project.optional-dependencies]
docs = [
  "sphinx < 9",  # Avoid breakage with major updates
  "myst-nb",     # Support for MyST syntax and notebooks
  "numpydoc",    # Support for NumPy's docstrings
  "sphinx-autoapi",    # To generate API documentation
  "sphinx-autobuild", # For interactive editing
  "sphinx-design",    # For additional web-components
  # ...
]
```

Project maintenance

Consider providing appropriate maintenance commands.

```toml
# pyproject.toml
[tool.hatch.envs.docs]
features = ["docs"]  # Add more groups if needed
python = "3.12"      # Freeze Python version for docs

[tool.hatch.envs.docs.scripts]
build = "sphinx-build docs/ docs/_build/html {args}"
serve = "sphinx-autobuild docs/ docs/_build/html {args}"
clean = "rm -rf docs/_build"
```

```
# console
hatch run docs:build
hatch run docs:serve
hatch run docs:clean
```

Automated deployments

Use any of readthedocs, GitLab pages or self-hosting, with a custom domain eventually.

```yaml
# .readthedocs.yaml
version: 2
python:
  install:
  - method: pip
    path: '.'
    extra_requirements:
    - docs  # Add more groups if needed
build:
  os: ubuntu-22.04
  tools:
    python: '3.12'
```

Take away

- Apply Diátaxis to raise your documentation's impact
  - Understand documentation needs and concerns
  - Adapt documentation structure and content accordingly
  - Avoid overlapping content, favour cross-referencing instead
- Use Sphinx with MyST for a modern and interactive experience
  - Sphinx with modern plugins provides a rich editing experience
  - MyST is expressive enough to fullfil most documentation needs
  - MyST notebooks are easier to keep track of and review
- Invest time and efforts in automation and maintenance setup
  - Treat your documentation as a separate project
  - Make your documentation builds reproducible

Merci.