OMNeT++ doc-builder

OMNeT++ developers

Nov 09, 2026

CONTENTS

1 Installation **3**
- 1.1 The workflow . . . 3
- 1.2 Enabling GitHub Pages . . . 4
- 1.3 What it publishes . . . 4
- 1.4 The workflow of this site . . . 4

2 A Sphinx Site **7**
- 2.1 The configuration . . . 7
- 2.2 This site as an example . . . 8

3 The C++ API **11**
- 3.1 In a Sphinx site . . . 11

4 PDF **13**
- 4.1 A PDF of a part of the site . . . 13
- 4.2 Links in the PDF . . . 13

5 Models That Build on Other Models **15**

6 Running Locally **17**

7 Versions and the Image **19**
- 7.1 The image . . . 19
- 7.2 Versions . . . 19

The action builds the documentation website of an OMNeT++ model project into a directory, and by default uploads it as the GitHub Pages artifact. These pages show how to set it up, from the NED reference alone to a Sphinx site with the C++ API and PDFs.

CHAPTER

ONE

INSTALLATION

The action needs no installation: a workflow of your model's repository uses it, and GitHub fetches it. This page publishes the NED reference of a model; *A Sphinx Site* adds your own pages.

1.1 The workflow

`.github/workflows/docs.yml` in your model repository:

```yaml
name: Documentation

on:
  push:
    branches: [master]
  workflow_dispatch:

concurrency:
  group: pages
  cancel-in-progress: false

jobs:
  build:
    runs-on: ubuntu-latest
    permissions:
      contents: read
    steps:
      - uses: actions/checkout@v7
      - uses: omnetpp/doc-builder@v1

  deploy:
    needs: build
    runs-on: ubuntu-latest
    permissions:
      pages: write
      id-token: write
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    steps:
      - id: deployment
        uses: actions/deploy-pages@v5
```

The deployment is a second job, because it needs permissions that the build does not.

1.2 Enabling GitHub Pages

Once, in the settings of the repository: **Settings** → **Pages** → **Build and deployment** → **Source: GitHub Actions**. The next run of the workflow publishes the site, at `https://<owner>.github.io/<repository>/`.

1.3 What it publishes

The NED reference of the project, in the repository root, in the look of the NED documentation of the Eclipse-based IDE (`ned-format: html-material`; `html-readthedocs` and `html` are the other choices). The NED folders of the project are taken from its `.nedfolders` file, like in the IDE.

Add `doxygen: true` to also publish the C++ API, under `api/`. The NED types' pages then link to their C++ classes ("C++ definition"):

```
- uses: omnetpp/doc-builder@v1
  with:
    doxygen: true
```

1.4 The workflow of this site

This site is deployed by the workflow below, from the `site/` directory of the doc-builder repository. It builds the site three times: the site itself, with the C++ API and a PDF, and two variants of it into subdirectories, so it uploads the Pages artifact itself.

```
# Builds the website of this repository (site/) with the action, the way a model project␣
↪uses it, and deploys
# it to GitHub Pages. Two variants of the site, in variants/, show what settings that␣
↪apply to the whole site
# change. The site documents the released action, so it is built on pushes to the␣
↪release branch.
name: Pages

on:
  push:
    branches: [v1]
  workflow_dispatch:

concurrency:
  group: pages
  cancel-in-progress: false

jobs:
  build:
    runs-on: ubuntu-latest
    permissions:
      contents: read
    steps:
      - uses: actions/checkout@v7

      - uses: omnetpp/doc-builder@v1
        with:
          project-dir: site
```

(continues on next page)

(continued from previous page)

```
          title: Sample Model
          sphinx-source: site/doc
          doxygen: true
          pdf: true
          fail-on-warnings: true
          upload-pages-artifact: false

      # The tabs open no menus
      - uses: omnetpp/doc-builder@v1
        with:
          project-dir: site
          title: Sample Model
          sphinx-source: site/doc
          doxygen: true
          sphinx-args: -t variant -D omnetpp_nav_menus=0
          fail-on-warnings: true
          output-dir: _site/variants/no-menus
          upload-pages-artifact: false

      # The NED reference alone, without a Sphinx project: what the quick start publishes
      - uses: omnetpp/doc-builder@v1
        with:
          project-dir: site
          title: Sample Model
          doxygen: true
          fail-on-warnings: true
          output-dir: _site/variants/ned-only
          upload-pages-artifact: false

      - uses: actions/upload-pages-artifact@v5
        with:
          path: _site

  deploy:
    needs: build
    runs-on: ubuntu-latest
    permissions:
      pages: write
      id-token: write
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    steps:
      - id: deployment
        uses: actions/deploy-pages@v5
```

CHAPTER

TWO

A SPHINX SITE

With `sphinx-source`, the site is built from your Sphinx project, and the NED reference (and the C++ API) become pages of it:

```
- uses: omnetpp/doc-builder@v1
  with:
    sphinx-source: doc
    doxygen: true
```

The action generates the NED reference as reStructuredText into `<sphinx-source>/ned` (`ned-dir`) and the C++ API into `<sphinx-source>/api` (`api-dir`), then runs `sphinx-build` in the Sphinx project. Your project refers to them from a toctree:

```
.. toctree::

   guide
   NED Reference <ned/index>
   C++ Reference <api/index>
```

and ignores them in git, since they are generated (`doc/.gitignore`):

```
/ned/
/api/
```

2.1 The configuration

The action does not change `conf.py`; it is yours. The image provides Sphinx, `sphinx_immaterial` and the OMNeT++ extensions (see ../config/theme and ../config/extensions). A `conf.py` that gives the site the look of the OMNeT++ sites:

```
project = 'My Model'

extensions = [
    'sphinx_immaterial',
    'omnetpp_immaterial',   # the look of the OMNeT++ sites
    'omnetpp_neddoc',       # the :ned: and :msg: roles, NED/MSG highlighting
    'omnetpp_doxylink',     # the :cppref: role (with doxygen: true)
]
html_theme = 'sphinx_immaterial'

doxylink = {'cppref': ('api/tagfile.xml', 'api/')}
```

`omnetpp_neddoc` finds the NED reference in `ned` (its `neddoc_dir` setting; change it together with `ned-dir`).

The NED reference and the C++ API have a page per type, too many for the navigation that Sphinx puts into every page. `omnetpp_immaterial` shows them in the sidebar as trees that the browser loads when the reader opens them: the NED reference by packages, kinds of types and files, the C++ API by namespaces and classes. The trees appear under the entries of `ned/index` and `api/index` in your toctree; open the NED Reference of this site to see them.

The pages of the site refer to NED and MSG types and to C++ classes with roles, and highlight NED, MSG and ini code (see ../config/writing).

2.2 This site as an example

The site/doc directory of the repository is the Sphinx project of this site. Its `conf.py`:

```
import os

project = 'doc-builder'
author = 'OMNeT++ developers'
copyright = '2026, OpenSim Ltd.'

extensions = [
    'sphinx_immaterial',
    'omnetpp_immaterial',   # the look of the OMNeT++ sites
    'omnetpp_neddoc',       # the :ned: and :msg: roles, linking to the NED reference in ned/
    'omnetpp_doxylink',     # the :cppref: role, linking to the C++ API
]

html_theme = 'sphinx_immaterial'
html_title = 'OMNeT++ doc-builder'
html_baseurl = 'https://omnetpp.org/doc-builder/'
html_theme_options = {
    'site_url': html_baseurl,
    'repo_url': 'https://github.com/omnetpp/doc-builder',
    'repo_name': 'omnetpp/doc-builder',
    'icon': {
        'repo': 'fontawesome/brands/github',
        'logo': 'material/book-open-page-variant-outline',
    },
    'globaltoc_collapse': False,
    'toc_title_is_page_title': True,
    'features': [
        'navigation.expand',
        'navigation.instant',
        'navigation.sections',
        'navigation.tabs',
        'navigation.top',
        'search.highlight',
        'toc.follow',
        'toc.sticky',
    ],
    'palette': [
        {
            'media': '(prefers-color-scheme: light)',
```

(continues on next page)

(continued from previous page)

```
            'scheme': 'default',
            'primary': 'indigo',
            'accent': 'deep-orange',
            'toggle': {'icon': 'material/lightbulb-outline', 'name': 'Switch to dark mode
↪'},
        },
        {
            'media': '(prefers-color-scheme: dark)',
            'scheme': 'slate',
            'primary': 'indigo',
            'accent': 'amber',
            'toggle': {'icon': 'material/lightbulb', 'name': 'Switch to light mode'},
        },
    ],
}

# Each tab shows one of the ways the tabs can be navigated (see config/navigation.rst)
omnetpp_nav_no_menu = ['Home']
omnetpp_nav_dropdown = ['Configuration']
omnetpp_nav_nowrap = ['/ned/', '/api/']
# The sample API is too small to reach omnetpp_nav_json_threshold; the tree is what a␣
↪large API would get
omnetpp_nav_json = ['api/index']

# Untracked files are not counted: the build's own output is among them.
omnetpp_commit = os.popen('git rev-parse --short=10 HEAD').read().strip()
if omnetpp_commit and os.popen('git status --porcelain --untracked-files=no').read():
    omnetpp_commit += '+'
omnetpp_commit = omnetpp_commit or None

# The doc-builder action generates the C++ API into api/, with its doxygen tag file
doxylink = {'cppref': ('api/tagfile.xml', 'api/')}

# The PDF is the guide alone: the NED reference and the C++ API are pages for the web.
latex_documents = [('guide/index', 'doc-builder.tex', 'OMNeT++ doc-builder', author,
↪'manual')]
```

To build the site locally while writing it, see *Running Locally*.

CHAPTER

THREE

THE C++ API

`doxygen: true` runs doxygen with a bundled configuration (that of the OMNeT++ API documentation, with doxygen-awesome) on the header files under `<project-dir>/src` (`doxygen-input`; generated `*_m.h` files are excluded). Undocumented classes and members are left out.

`doxyfile` replaces the bundled configuration with yours. Either way, the action overrides the output settings (`OUTPUT_DIRECTORY`, `HTML_OUTPUT`, `GENERATE_TAGFILE`, ...), and, in a Sphinx site, those that the wrapping of the pages into the site needs (`GENERATE_TREEVIEW = NO`, `DISABLE_INDEX = NO`, `SEARCHENGINE = NO`). Paths in the doxyfile are relative to the working directory, i.e. the repository root.

3.1 In a Sphinx site

The doxygen pages are wrapped into Sphinx pages by `doxysphinx`, like in the OMNeT++ documentation, without doxygen's own title and navigation bar, and `api/index` is the class list, titled "<title> C++ Reference". Only the overview pages (class hierarchy, class index, namespace list) and the topic pages are in the site navigation, under `api/index`; the class, namespace and file pages are not, so that a large API does not add an entry per class to the navigation of every page. The tree of the C++ API in the sidebar lists them instead.

Doxygen comments can contain reStructuredText between `\rst` and `\endrst`.

The C++ Reference of this site is the API of the sample model, four classes in two namespaces.

CHAPTER

FOUR

PDF

With `pdf: true`, the action also builds the Sphinx project with `sphinx-build -M latexpdf` (the image has TeX Live), and copies the PDFs into the root of the site, where the pages can link to them. Without `sphinx-source` there is no PDF: `opp_docgen` writes none.

4.1 A PDF of a part of the site

What goes into a PDF is up to `latex_documents` in `conf.py`: each entry is a PDF, whose root document is the start of its toctree. The NED reference and the C++ API are pages for the web, so let the root be a document whose toctree does not include them, e.g. that of a guide:

```
latex_documents = [('guide/index', 'my-model.tex', 'My Model Guide', author, 'manual')]
```

The PDF of this site is built this way from its Getting Started pages, the toctree of `guide/index`:

```
latex_documents = [('guide/index', 'doc-builder.tex', 'OMNeT++ doc-builder', author,
↪'manual')]
```

4.2 Links in the PDF

Links to pages that are not in the PDF (e.g. from the guide to the NED reference) become their text. A `:cppref:` link to the C++ API on the site by a relative path (`doxylink = {'cppref': ('api/tagfile.xml', 'api/')}`) cannot work in a PDF either: it also becomes its text, with a warning, which fails the build with `fail-on-warnings`. Keep `:cppref:` out of the PDF's documents, as this site does, or give doxylink the API's URL, e.g. the published site's, and the PDF links there:

```
doxylink = {'cppref': ('api/tagfile.xml', 'https://example.org/my-model/api/')}
```

CHAPTER

FIVE

MODELS THAT BUILD ON OTHER MODELS

A model that uses the NED types of another one (e.g. INET) needs the other's NED files, to resolve the types it uses. They are not documented, but linked to, if you tell where their documentation is:

```
- uses: actions/checkout@v7
- uses: actions/checkout@v7
  with:
    repository: inet-framework/inet
    ref: v4.6.0
    path: inet
- uses: omnetpp/doc-builder@v1
  with:
    ned-path: inet/src
    external-docs: inet=https://inet.omnetpp.org/docs/latest/api/types
```

`external-docs` maps NED packages to the documentation of their types; a type's page is `<url>/<qualified name>.html`, so the URL above links to the NED reference of the latest INET release. The pages of message types are named differently, which `opp_docgen` can only know from the tag files of the dependency's documentation (`msgtags.xml`); INET does not publish them, so the links to INET's message types do not work.

In the pages of a Sphinx site, `:ned:` and `:msg:` link only to the project's own types: the name of a dependency's type renders as code, with a `neddoc.unknown` warning (`suppress_warnings = ['neddoc.unknown']` in `conf.py` silences it). Link to the dependency's page with an ordinary link instead.

Remember to exclude the checkout from your own documentation if it is under your project directory and you have no `.nedfolders`, e.g. with `docgen-args: -x inet`.

The sample model of this site has no dependencies, so its workflow does not do this.

CHAPTER

SIX

RUNNING LOCALLY

`doc-builder.sh` is the whole action; it runs the same build on your machine, with podman (or docker: `ENGINE=docker`). From the repository root of your model, with a checkout of the doc-builder repository next to it:

```
../doc-builder/doc-builder.sh --sphinx-source doc --doxygen
```

It takes the inputs of the action as options (`--help` lists them), and builds into `_site`, which you can open in a browser or serve with:

```
python3 -m http.server -d _site
```

Pull the image again to update it:

```
podman pull ghcr.io/omnetpp/doc-builder:latest
```

Inside a container that is the image (`ENGINE=none`), it runs the tools directly.

This site, from the root of the doc-builder repository:

```
./doc-builder.sh --project-dir site --title "Sample Model" \
    --sphinx-source site/doc --doxygen --pdf
```

CHAPTER

SEVEN

VERSIONS AND THE IMAGE

7.1 The image

The image is built from the doc-builder repository and released with the action: by default, the action runs in the image of its own version, e.g. `omnetpp/doc-builder@v1` in `ghcr.io/omnetpp/doc-builder:v1`, which is built from the `v1` branch, like the action. To keep the documentation build stable, use a full version of the action (`omnetpp/doc-builder@v1.2.0`), or pin an image tag with `image-tag` (see the tags). `latest` is the newest image.

The image is `linux/amd64` only, so the action runs on Linux runners (`ubuntu-latest`), with podman or docker.

If you need a build that the action does not do, run your own commands in the image, with the same tools:

```yaml
jobs:
  build:
    runs-on: ubuntu-latest
    container: ghcr.io/omnetpp/doc-builder:latest
    steps:
      - uses: actions/checkout@v7
      - run: opp_docgen --format=rst -o doc/ned . && sphinx-build -b html doc _site
      - uses: actions/upload-pages-artifact@v5
        with:
          path: _site
```

7.2 Versions

The image and the action share the versions of the repository. A major version is a branch (`v1`), and is what a workflow normally uses: it gets fixes and new features, but no changes that break existing workflows. Those come with a new major version. Releases are tags `vMAJOR.MINOR.PATCH` on it; use one, or a commit, to pin an exact version.

Each version is also an image tag: the image built from the `v1` branch is `:v1`, that of a release `:v1.2.0`, and the action runs in the image of its own version by default.