The Book Writer Manual

Contents

How to contribute

Format and Style

This book describes the features of our documentation system and the procedures to update it.

Overview

Where the documentation is

The documentation is broken down into a collection of books, with each book contained in a separate repository called `book-[name]`:

For example:

- **`book-devmanual-docs`** (this book)
- **`book-opmanual-duckiebot`**

Documentation format

The documentation is written as a series of small files in Markdown format.

It is then processed by a series of scripts to create publication-quality PDF and an online HTML version.

You can find all these artifacts produced at the site **`https://docs.duckietown.com`**.

The simple way: Github

The simplest way to contribute to the documentation is to hover the GitHub icon at the top the documentation page we want to edit, and then press “suggest edit” from the dropdown menu as shown in the image below.

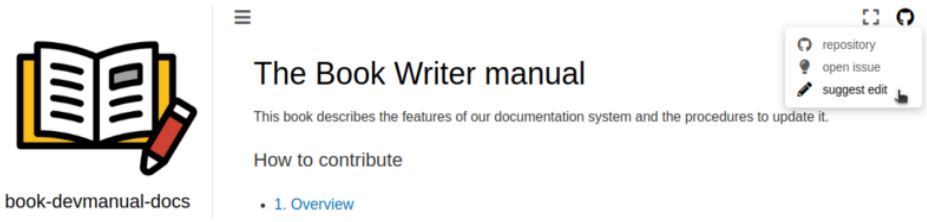


Fig. 1 Click on “suggest edit” button from the GitHub dropdown menu.

This will take you to the text editor on GitHub. One can make and commit the edits in only a few seconds.

If you are in the Duckietown organization, then you can edit the files directly and commit to a personal branch for review (more on this later). If you are not a member of Duckietown, you will be asked to fork the repository first, follow the instructions shown on GitHub to do so.

Let us try this out and perform an edit:

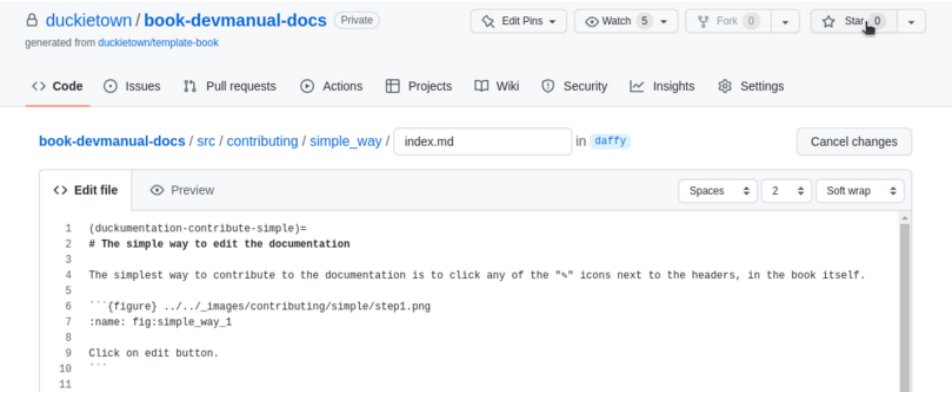


Fig. 2 Perform the edits you had in mind.

Once done, check the temporary outcome by clicking on `Preview changes`. Note that not all functionalities are visible by the preview. For significant changes to the documentation, refer to The right way: Local editor.

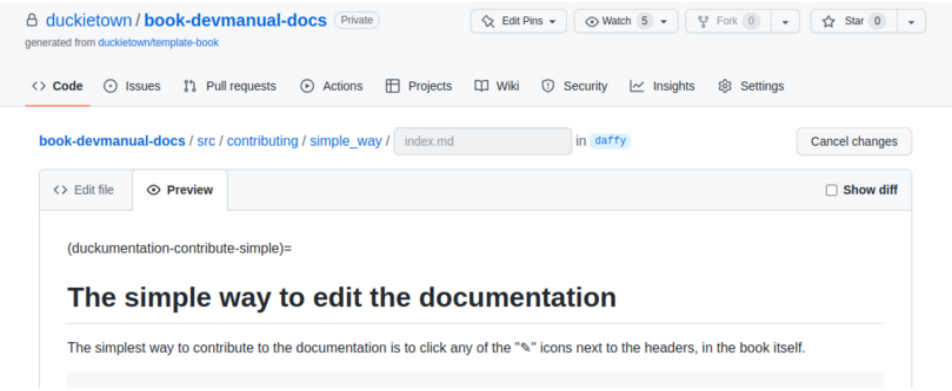


Fig. 3 A preview of the changes.

At the end of the webpage, use the form provided to describe your changes. Start by choosing “Create a new branch for this commit and start a pull request”, this will give you the opportunity to submit your changes for review before they go live. Then fill in the commit message and description fields before confirming by clicking on the “Propose changes” button (Fig. 4).

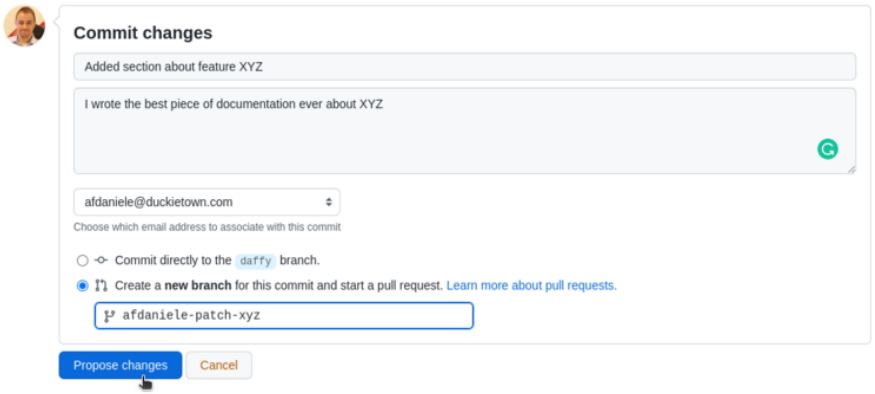


Fig. 4 Describe and commit.

This will take you to the Pull Request creation page (Fig. 5). Add any further details you want to share with the reviewers in the Pull Request description text area.

Complete by clicking on the `Create pull request` button (Fig. 5).

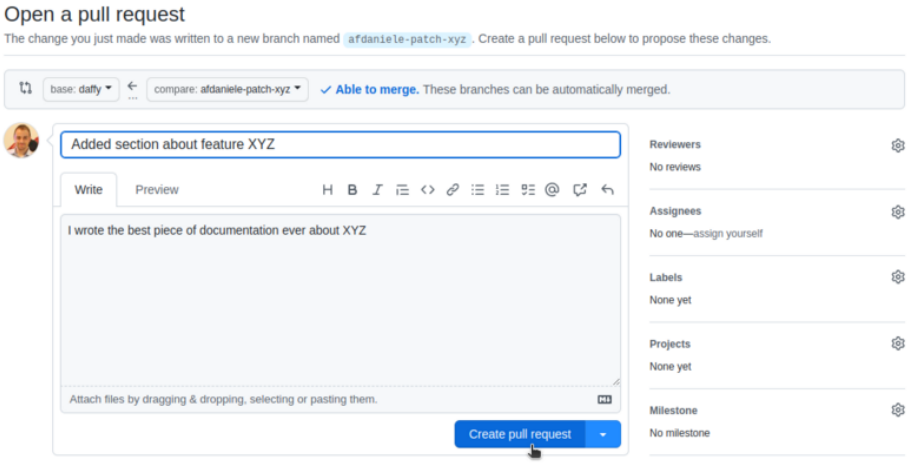


Fig. 5 Create a pull request.

The right way: Local editor

This section describes the workflow to edit the documentation for one single book.

In a nutshell:

- You *fork* the repos to your GitHub account.
- You compile locally using a Docker container (no installation necessary).
- You contribute by opening a pull request.

Workflow

GitHub setup

We assume that you have set up a GitHub account with working public keys.

See: Basic SSH config.

See: Key pair creation.

See: Adding public key on GitHub.

Install Docker

Before you start, make sure that you have installed Docker.

Install the Duckietown Shell

Install the Duckietown Shell using these instructions.

Fork the `book-[name]` repository on GitHub

Navigate to the book repository page on GitHub, and click on the Fork button at the top-right corner of the page.

This will create a new repository on your account that is linked to the original one.

Checkout your fork locally

Check out the forked repository locally.

Do your edits

Do your edits on your local copy of the repository. The source files are in the directory `src/`.

Images are stored in the directory `src/_images`, while `CSS` and `JS` files can be dropped inside the directory `src/_static` and will be automatically loaded.

Compile HTML

Compile using the `dts docs` commands in the Duckietown Shell:

```
dts docs build
```

Clean up artifacts and build cache with the command,

```
dts docs clean
```

View the HTML

Once built, the book will be exported as HTML inside the directory `html/`. Open the file `html/index.html` to start. Make sure that your changes look the way you want them to.

Compile PDF

Compile the book into a PDF file using the command,

```
dts docs build --pdf
```

View the PDF

Once built, the book will be exported as PDF inside the directory `pdf/`. Open the file `pdf/book.pdf` to start. Make sure that your changes look the way you want them to.

Commit and push

Commit and push as you would do normally.

> **Attention**
>
> You need to be part of the **`Developers - Docs`** team on GitHub to be able to push changes to the documentation repositories. Ask your supervisor if you don't have access.

Make a pull request

Create a pull request to the original repository.

Publish artifacts directly

While it is recommended to use Continuous Integration (CI) systems (e.g., Jenkins, CircleCI) to perform automatic builds and deployments of the documentation, you can decide to push your local artifacts to the corresponding HTTP server. You can do so by running the following command,

```
dts docs publish [DNS]
```

where `[DNS]` is the hostname of the documentation website to push the artifacts to, e.g., `docs.duckietown.com`.

> **Note**
>
> This is only allowed on staging servers, e.g., `staging-docs.duckietown.com`. Only Jenkins can publish to production.

> **Note**
>
> This is a shortlisted version of the MyST syntax cheat sheet.

MyST syntax cheat sheet

Headers

Syntax	Example	Result
`# Heading level 1` `## Heading level 2` `### Heading level 3` `#### Heading level 4` `##### Heading level 5` `###### Heading level 6`	`# MyST syntax cheat sheet`	Level 1-6 headings, denoted by number of `#`

Target headers

Syntax	Example	Result
`(target_header)=`	`(myst_cheatsheet)=` `# MyST Cheat Sheet`	See below how to reference target headers.

Referencing target headers

```
[](myst_cheatsheet)
```

You can specify the text of the target:

```
[MyST syntax lecture](myst_cheatsheet)
```

Quote

Syntax	Example	Result
`> text`	`> this is a quote`	quoted text

Thematic break

Syntax	Example	Result
`---`	`This is the end of some text.` `---` `## New Header`	Creates a horizontal line in the output

Line comment

Syntax	Example	Result
`% text`	`a line` `% a comment` `another line`	See Comments for more information.

Block break

Syntax	Example	Result
`+++`	`This is an example of` `+++ {"meta": "data"}` `a block break`	This is an example of a block break

HTML block

Syntax	Example	Result
`<tagName> text </tagName>`	`<p> This is a paragraph </p>`	This is a paragraph

Links

Syntax	Example	Result
`[text](target)`	`[Jupyter Book]` `(https://jupyterbook.org)`	Jupyter Book
`[text](relative_path)`	`[Another page]` `(../style/index)`	Another page
`<target>`	`<https://jupyterbook.org>`	https://jupyterbook.org
`[text][key]`	`[Jupyter Book][intro_page]` `[intro_page]:` `https://jupyterbook.org`	Jupyter Book

Lists

Ordered list

Example	Result
`1. First item` `2. Second item` `1. First sub-item`	1. First item 2. Second item 1. First sub-item
`1. First item` `2. Second item` `* First sub-item`	1. First item 2. Second item - First subitem

Unordered list

Example	Result
`* First item` `* Second item` `* First subitem`	- First item - Second item - First subitem
`* First item` `1. First subitem` `2. Second subitem`	- First item 1. First subitem 2. Second subitem

Tables

Syntax	Example	Result
`| a | b |` `| :--- | ---: |` `| c | d |`	`| Training | Validation |` `| :---------- | ---------: |` `| 0 | 5 |` `| 13720 | 2744 |`	**Training** **Validation** 0 5 13720 2744
`` ```{list-table} Table title` `:header-rows: 1` `:name: label-to-reference`  `* - Col1` `- Col2` `* - Row1 under Col1` `- Row1 under Col2` `* - Row2 under Col1` `- Row2 under Col2` ````	```` ```{list-table} This table title` `:header-rows: 1` `:name: example-table`  `* - Training` `- Validation` `* - 0` `- 5` `* - 13720` `- 2744` ````	**Training** **Validation** 0 5 13720 2744 *Table 1* This table title

Referencing tables

> **Note**
>
> In order to reference a table, you must add a label to it. To add a label to your table simply include a `:name:` parameter followed by the label of your table. In order to add a *numbered reference*, you must also include a table title. See example above.

Syntax	Example	Result
`{numref}`label``	``{numref}`example-table` is an example.``	Table 1 is an example.
`[text](label)`	``This is an unnumbered ref: [table](example-table).``	This is an unnumbered ref: table.

Tabs

Tabs can be used in several ways:

1. At the page level to enclose instruction versions related to different releases (for example, to separate the DB19 and DB21 assembly instructions).
2. Within pages to divide duplicate content with tab based.
3. Nested within other components such as a list.

> **Note**
>
> Related content that does not include some duplication should be shown in a table rather than a tab to prevent hidden text.

Example	Result

````
````{tab-set}

```{tab-item} DB19
This is example content for the DB19
```

```{tab-item} DB21
This is example content for the DB21
```
````
````

DB19 DB21

This is example content for the DB19

Admonitions

Syntax	Result
``{note} Use note directives for basic highlighting. ```	**Note** Use note directives for basic highlighting.
``{warning} Use warnings for situations that might cause harm, but can be fixed. ```	**Warning** Use warnings for situations that might cause harm, but can be fixed.
``{tip} A tip is a useful suggestion for the reader. ```	**Tip** A tip is a useful suggestion for the reader.
``{attention} This directive should be used to highlight particularly tricky steps. ```	**Attention** This directive should be used to highlight particularly tricky steps.
``{danger} Used for situations that might cause irreparable harm (to people or robots). ```	**Danger** Used for situations that might cause irreparable harm (to people or robots).
``{seealso} Used for external links (to third-party websites or other documents). ```	**See also** Used for external links (to third-party websites or other documents).

All above specific admonitions are specific pre-made directives. You could make an admonition with custom title and class with the example below.

``{admonition} Title text ```	```{admonition} General admonition content ```	**General admonition** content
``{admonition} Title :class: warning text ```	```{admonition} This is a title :class: warning A custom admonition ```	**This is a title** A custom admonition

Icons

Icons are provided by the font-awesome project. The complete list of icons available can be found here.

Syntax	Example	Result
``{icon} <icon-id> ```	```{icon} ice-cream ```	

Figures

Syntax	Example	Result
``{figure} ./path/to/figure.jpg :name: label caption ```	```{figure} ../../_images/duckietown.jpeg :width: 50px :name: figure-example-2 Here is my figure caption! ```	 ***Fig. 6*** Here is my figure caption!
``{image} ./path/to/figure.jpg :name: label ```	```{image} ../../_images/duckietown.jpeg :scale: 20% :align: center :name: image-example ```	
![alt-text] (path/to/image)	![] (https://tinyurl.com/39ewhkab)	

Framed figures

Use the `:class: framed` parameter to add a border around the image.

Syntax	Example	Result
``{figure} ./path/to/figure.jpg :class: framed ```	```{figure} ../../_images/duckietown.jpeg :width: 50px :class: framed ```	

> **Note**
>
> - Content/caption is not permitted for *image*s, but only available for *figure*s.
> - Settings are not available with `![alt-text](path/to/image)` format

Referencing figures

Syntax	Example	Result
`{numref}`label``	`{numref}`figure-example-2`is a figure example.`	Fig. 6 is a figure example.
`{numref}`text %s <label>``	`{numref}`Figure %s <figure-example-2>` is an example.`	Figure 6 is an example.
`[text]<label>`	`This [figure](figure-example-2) is an example.`	This figure is an example.

Referencing images

Syntax	Example	Result
`[text](label)`	`This [image](image-example) is an example.`	This image is an example.

Videos

Videos can be referenced using the following methods:

1. `vimeo` - When possible, video content should be added to the Vimeo account and formatted with the custom Duckietown `vimeo` directive.
2. `videoembed` - For other video content accessible via a web link, use the `videoembed` directive. All `iframe` attributes are available mimicking the `:alt:` parameter syntax below.
3. `video` - For videos stored locally to the book project (this is not recommended), use the `video` directive. All `iframe` attributes are available mimicking the `:alt:` parameter syntax below.

Referencing Vimeo videos

Syntax	Example	Result
`` ```{vimeo} video-id :alt: alt text ``` ````	```` ```{vimeo} 527022343 :alt: alt text ``` ````	

Referencing web videos

Syntax	Example	Result
`` ```{video} embed_link :alt: alt text ``` ````	```` ```{video} https://www.youtube.com/embed/mXH1u885bn8 :alt: alt text ``` ````	

Referencing local videos

This is not recommended - please host your video content on Vimeo or another online service rather than in the book project. If absolutely necessary, you can include local videos with custom formatting using the `video` directive.

Supported file types: `.mp4`, `.ogm`, `.ogv`, `.ogg`, `.webm`.

Syntax	Example	Result
`` ```{video} file_path :alt: alt ``` ````	```` ```{video} ../assets/videos/my_video.mp4 :alt: alt text ``` ````	

Math

Syntax	Example	Result
Inline	`This is an example of an inline equation $z=\sqrt{x^2+y^2}$.`	This is an example of an inline equation $z = \sqrt{x^2 + y^2}$.
Math blocks	`This is an example of a math block` `$$` `z=\sqrt{x^2+y^2}` `$$`	This is an example of a math block $$z = \sqrt{x^2 + y^2}$$
Math blocks with labels	`This is an example of a math block with a label` `$$` `z=\sqrt{x^2+y^2}` `$$ (eq-label)`	This is an example of a math block with a label $$z = \sqrt{x^2 + y^2}$$ (1)

Referencing math directives

Syntax	Example	Result
`[](label)`	`Check out equation [](eq-label).`	Check out equation (1).

Code

In-line code

Example:

```
Wrap in-line code blocks in backticks: `boolean example = true;`.
```

Result:

Wrap in-line code blocks in backticks: `boolean example = true;`.

Code and syntax highlighting

Example:

````
```python
note = "Python syntax highlighting"
print(node)
```
````

or

````
```none
No syntax highlighting.
```
````

Result:

```
note = "Python syntax highlighting"
print(node)
```

or

```
No syntax highlighting.
```

Executable code

> **Warning**
>
> Make sure to include this front-matter YAML block at the beginning of your `.ipynb` or `.md` files.
>
> ```
> ---
> jupytext:
>   formats: md:myst
>   text_representation:
>     extension: .md
>     format_name: myst
> kernelspec:
>   display_name: Python 3
>   language: python
>   name: python3
> ---
> ```

Example:

````
```{code-cell} ipython3
note = "Python syntax highlighting"
print(note)
```
````

Result:

```
note = "Python syntax highlighting"
print(note)
```

```
Python syntax highlighting
```

Tags

See the tags section on Jupyter Books documentation. For formatting the code cells.

Gluing variables

Example:

````
```{code-cell} ipython3
from myst_nb import glue
my_variable = "here is some text!"
glue("glued_text", my_variable)
```

Here is an example of how to glue text: {glue:}`glued_text`
````

Result:

```
from myst_nb import glue
my_variable = "here is some text!"
glue("glued_text", my_variable)
```

```
'here is some text!'
```

Here is an example of how to glue text: `'here is some text!'`

Gluing numbers

Example:

````
```{code-cell} ipython3
from myst_nb import glue
import numpy as np
import pandas as pd

ss = pd.Series(np.random.randn(4))
ns = pd.Series(np.random.randn(100))

glue("ss_mean", ss.mean())
glue("ns_mean", ns.mean(), display=False)
```

Here is an example of how to glue numbers: {glue:}`ss_mean` and {glue:}`ns_mean`.
````

Result:

```
from myst_nb import glue
import numpy as np
import pandas as pd

ss = pd.Series(np.random.randn(4))
ns = pd.Series(np.random.randn(100))

glue("ss_mean", ss.mean())
glue("ns_mean", ns.mean(), display=False)
```

```
/usr/local/lib/python3.8/dist-packages/pandas/core/computation/expressions.py:20:
UserWarning: Pandas requires version '2.7.3' or newer of 'numexpr' (version
'2.7.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
```

```
0.540809882310568
```

Here is an example of how to glue numbers: `0.540809882310568` and `0.10989813221697081`.

Gluing visualizations

Example:

````
```{code-cell} ipython3
from myst_nb import glue
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 200)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y, 'b-', linewidth=2)

glue("glued_fig", fig, display=False)
```

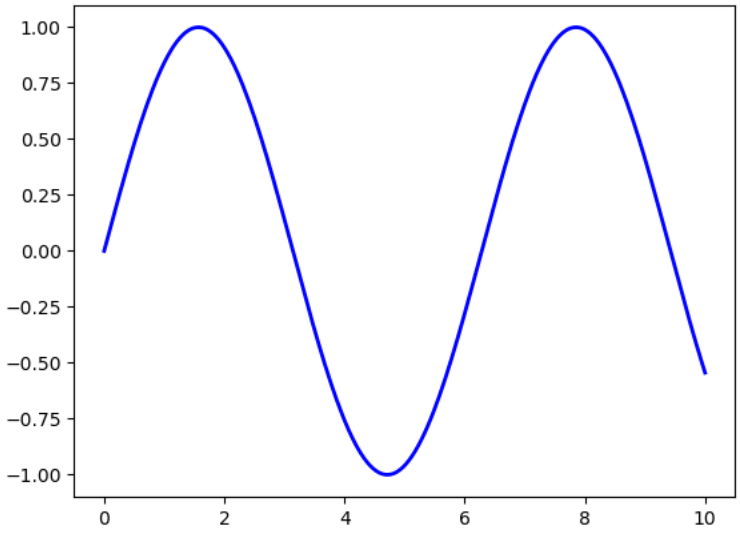
This is an inline glue example of a figure: {glue:}`glued_fig`.
This is an example of pasting a glued output as a block:
```{glue:} glued_fig
```
````

Result:

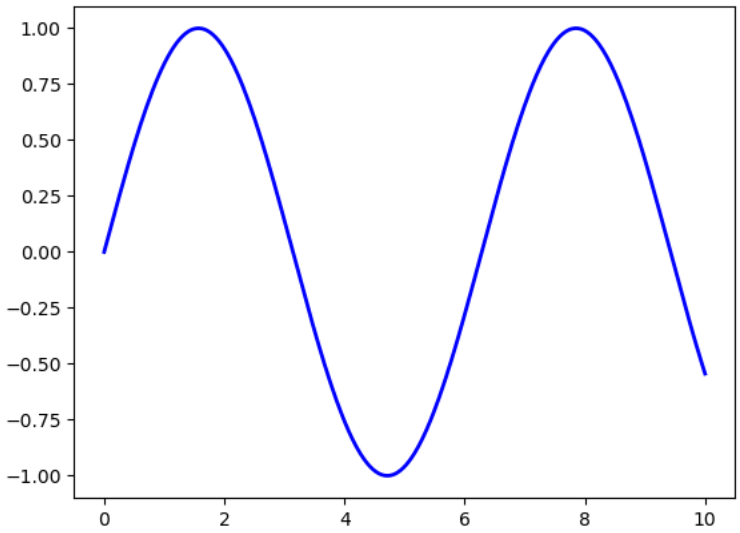
```
from myst_nb import glue
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 200)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y, 'b-', linewidth=2)

glue("glued_fig", fig, display=False)
```



This is an inline glue example of a figure:  . This is an example of pasting a glued output as a block:



Gluing math

Example:

````
```{code-cell} ipython3
import sympy as sym
x, y = sym.symbols('x y')
z = sym.Function('z')
z = sym.sqrt(x**2+y**2)
glue("example_eq", z, display=False)
```

To glue a math equation try
```{glue:math} example_eq
:label: glue-eq-example
```
````

Result:

```
import sympy as sym
x, y = sym.symbols('x y')
z = sym.Function('z')
z = sym.sqrt(x**2+y**2)
glue("example_eq", z, display=False)
```

To glue a math equation try:

$$\sqrt{x^2 + y^2} \qquad ()$$

Reference documents

Syntax	Example	Result
`[] (path/to/document)`	`Ref to [](../style/index)`	Ref to Style guide
`[text] (path/to/document)`	`See [here](../style/index) for more information.`	See here for more information.

Cross-book references

The cross-book referencing is achieved using the intersphinx plugin for sphinx. For a cross-book reference, you need to know the book name and the label defined within that book.

> **Note**
>
> All books hosted on `docs.duckietown.com` are automatically made available to be linked from any other Duckietown book. The book name is the repository name.

Example

```
Link to a [page on another book](book-opmanual-duckiebot:duckiebot-boot).
```

Link to a page on another book.

How to configure

The mapping between book names and their remote location is defined in the `_config.yml` file. For example, we can add a link to the book `jupyter-book-docs` as follows,

```
sphinx:
  ...
  config:
    intersphinx_mapping:
      jupyter-book-docs:
        - "https://jupyterbook.org/en/stable"
        - null
```

Advanced: list all labels for a given book

To conveniently find the available labels in other books, a utility comes with jupyter-book installation. Take the above linked book for example:

```
python -m sphinx.ext.intersphinx
https://jupyterbook.org/en/stable/objects.inv
```

Or, use it in a script (example outcomes provided below)

```
from sphinx.ext.intersphinx import inspect_main
inspect_main(["https://jupyterbook.org/en/stable/objects.inv"])
```

Footnotes

Syntax	Example	Result
`[^ref]` `[^ref]: Footnote text`	`This is a footnote reference. [^myref]` `[^myref]: This **is** the footnote definition.`	This is a footnote reference.[1]

Note

Footnotes are displayed at the very bottom of the page.

Troubleshooting

Troubleshooting cards can be created using the `{trouble}` directive.

Syntax	Example
`` ```{trouble} symptom here --- resolution here ``` ````	**Troubleshooting** SYMPTOM: I do not see a camera image. RESOLUTION: Make sure the camera cable is plugged in.

Requirements

Requirements/outputs cards can be created using the `{needget}` directive.

Syntax	Example
`` ```{needget} * Requirement 1 * Requirement 2 --- * Output 1 ``` ````	What you will need: • Duckie • Robot What you will get: • Duckiebot

Tests

You can use the Test / What to Expect card (`testexpect`) to define checkpoints after book instructions.

Syntax	Result
`` ```{testexpect} Test --- Expect ``` ````	Test: `pip3 --version` Expected Result: This command should output a version number for the `pip3` package.

ToDos

You can drop **ToDos** throughout the documentation using the `{todo}` directive. ToDos are rendered only on the staging documentation, they are hidden in production.

Syntax	Example
`` ```{todo} todo message here ``` ````	TODO todo message here

Citations

> **Note**
>
> Make sure you have a reference bibtex file. And it is included in the `_config.yml`, under `bibtex_bibfiles` section.

Syntax	Example	Result
{cite}`mybibtexcitation`	An example citation {cite}`tani2016`.	An example citation [TPZ+17].

And, at the bottom of the page, include the list of references:

```
```{bibliography}
:filter: docname in docnames
```
```

[TPZ+17] Jacopo Tani, Liam Paull, Maria T. Zuber, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. Duckietown: an innovative way to teach autonomy. In Dimitris Alimisis, Michele Moro, and Emanuele Menegatti, editors, *Educational Robotics in the Makers Era*, 104–121. Cham, 2017. Springer International Publishing.

[1] This **is** the footnote definition.

Style guide

This chapter describes the style guide for our documentation. We will cover the conventions for writing the technical documentation.

Organization

The documentation is divided into **books**, **parts** (labeled `part:`), **chapters** (labeled `chapter:`), and **sections** (labeled `sec:`).

The structure of each book is stored inside the `src/_toc.yml` file.

General guidelines for technical writing

The following holds for all technical writing.

- The documentation is written in correct English.
- The words “should” and “must” are not interchangeable, they have precise meanings;[1]
- “Please” is unnecessary in technical documentation;

> **Wrong**
>
> “Please remove the SD card.”

> **Better**
>
> “Remove the SD card.”

- Do not use colloquialisms or abbreviations;

> **Wrong**
>
> “The pwd is `ubuntu`.”

> **Better**
>
> “The password is `ubuntu`.”

- Python is capitalized when used as a name;

> **Wrong**
>
> “If you are using python…”

> **Better**
>
> “If you are using Python…”

- Do not use contracted forms;

> **Wrong**
>
> it’s

> **Better**
>
> it is

- Do not use emojis;
- Do not use **ALL CAPS**;
- Make infrequent use of **bold statements**;
- Do not use exclamation points;

Style guide for the Duckietown documentation

- The English version of the documentation is written in American English;

> **Incorrect**
>
> behaviour

> **Correct**
>
> behavior

- All the filenames and commands must be enclosed in code blocks using Markdown backticks;

> **Incorrect**
>
> “Edit the ~/.ssh/config file using nano.”

> **Correct**
>
> “Edit the `~/.ssh/config` file using `nano`.”

- `Ctrl` - `C` , `ssh`, etc. are not verbs;

> **Incorrect**
>
> “ `Ctrl` - `C` from the command line.”

> **Correct**
>
> “Use `Ctrl` - `C` from the command line.”

- Subtle humor and puns about duckies are encouraged.

Do make use of the necessary complexity to convey your message, but do not hide behind overly complex language to disguise flaws. Remember Einstein’s quote:

> You don’t really understand something unless you can explain it to your grandmother.

> **Examples**
>
> provide → give
>
> query → question
>
> in order to → to
>
> utilize → use

Frequently misspelled words

- “Duckiebot” is always capitalized.
- Use “Raspberry Pi”, not “PI”, “raspi”, etc.
- These are other words frequently misspelled:
  5 GHz WiFi

Other conventions

When the user must edit a file, just say: “edit `/this/file`”.

Writing down the command line for editing, like the following:

```
vi /this/file
```

is too much detail. Only specify the editor to use if the task at hand requires functionalities that are only available on a specific editor.

Troubleshooting sections

Write the documentation as if every step succeeds.

Then, at the end, make a “Troubleshooting” section.

Organize the troubleshooting section as a list of symptom/resolution.

The following is an example of a troubleshooting section.

Troubleshooting

Use the `{trouble}` directive to declare troubleshooting steps. For example,

Troubleshooting

SYMPTOM	This strange thing happens.
RESOLUTION	Maybe the camera is not inserted correctly. Remove and reconnect.

[1] These meanings are explained in this document.