



Duckumentation documentation

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

Contents

Part A - Contributing to the documentation	3
○ Unit A-1 - Overview.....	4
○ Unit A-2 - The simple way to edit the documentation	5
○ Unit A-3 - Second method: local editing+docker	8
○ Unit A-4 - Troubleshooting for Duckumentation.....	13
○ Unit A-5 - Markdown editors.....	15
Part B - Markduck format.....	17
○ Unit B-1 - Basic Markduck guide	18
○ Unit B-2 - Special paragraphs	27
○ Unit B-3 - LaTeX constructs.....	32
○ Unit B-4 - Embedding videos.....	39
○ Unit B-5 - Troubleshooting	40
○ Unit B-6 - Knowledge Boxes	42
Part C - Documentation style guide.....	45
○ Unit C-1 - Style guide.....	46
Part D - Slides.....	50
○ Unit D-1 - Creating slides.....	51
○ ? Making slides.....	52

PART A

Contributing to the documentation

This part describes the workflow for contributing to our documentation.

Contents

Unit A-1 - Overview.....	4
Unit A-2 - The simple way to edit the documentation.....	5
Unit A-3 - Second method: local editing+docker.....	8
Unit A-4 - Troubleshooting for Duckumentation.....	13
Unit A-5 - Markdown editors	15

UNIT A-1

Overview

1.1. Where the documentation is

The documentation is contained in a series of repositories called `docs-short name`:

For example:

- `docs-duckumentation` (this book)
- `docs-AIDO`

1.2. 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 on-line HTML version.

You can find all these artifacts produced at the site <https://docs.duckietown.org>.

UNIT A-2

The simple way to edit the documentation

The simplest way to contribute to the documentation is to click any of the “✎” icons next to the headers, in the book itself.

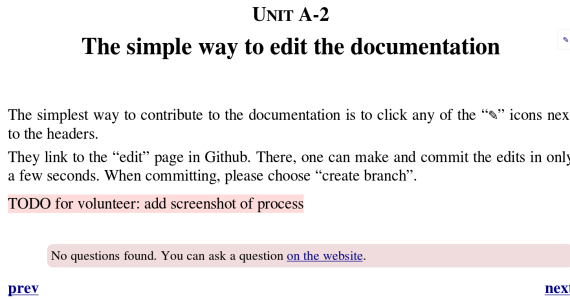


Figure 2.1. Click on edit button.

They link to the “edit” page in Github. One can make and commit the edits in only a few seconds.

If you are in the Duckietown organization, then you can just “edit” the page then and there. Otherwise, you need to “fork” the repo. Don’t worry, it’s a one click process.

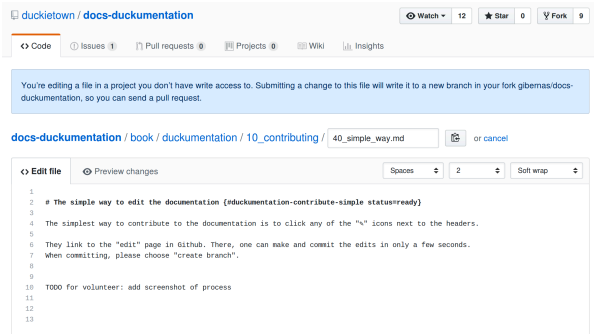


Figure 2.2. You will land on GitHub

Do your edits where appropriate.

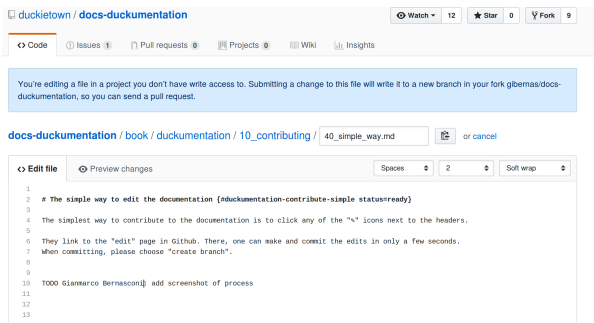


Figure 2.3. Editing the docs.

You can check the outcome by clicking on `Preview changes`, note that not all functionalities are visible by the preview. For large changes refer to Unit A-3 - Second method: local editing+docker.

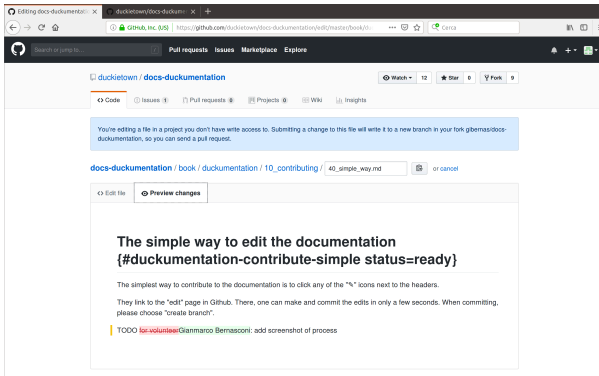


Figure 2.4. A preview of the changes.

Then after your edits, in the bottom part of the webpage, describe your commit and click on `Propose file change`. When committing, please choose "create branch".

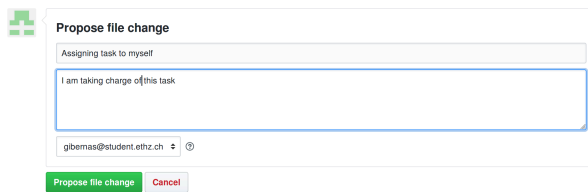


Figure 2.5. Describe and commit.

Then, click on `Create pull request`, as you probably don't have rights to push directly.

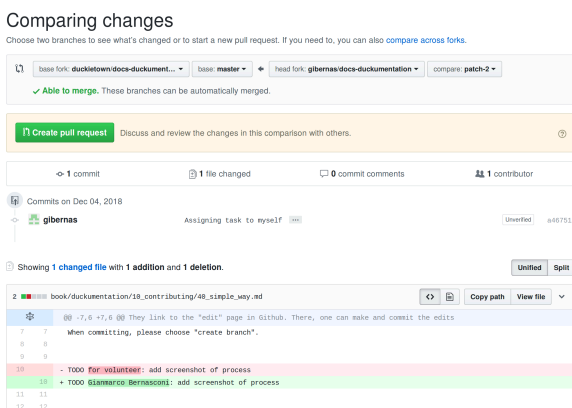


Figure 2.6. Create a pull request.

Again you need to confirm that you want to open the pull request.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base fork: duckietown/docs-duckument... | base: master | head fork: gibernaa/docs-duckumentation | compare: patch-2

✓ Able to merge. These branches can be automatically merged.

Assigning task to myself

Write | Preview

I am taking charge of this task

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Allow edits from maintainers. [Learn more](#) Create pull request

Figure 2.7. Confirm a pull request.

UNIT A-3

Second method: local editing+docker

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.

3.1. Workflow

1) Github setup

We assume that you have setup a Github account with working public keys.

- Basic SSH config ([unknown ref software_reference/github-access](#))

previous **warning** next (1 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#software_reference/github-access'.

Location not known more precisely.

Created by function n/a in module n/a.

- Key pair creation ([unknown ref software_reference/howto-create-key-pair](#))

previous **warning** next (2 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#software_reference/howto-create-key-pair'.

Location not known more precisely.

Created by function n/a in module n/a.

- Adding public key on Github ([unknown ref software_reference/github-access](#))

previous **warning** next (3 of 18) index

warning


```
I will ignore this because it is an external link.
```

```
> I do not know what is indicated by the link '#software_reference/github-access'.
```

Location not known more precisely.

Created by function n/a in module n/a.

2) Install Docker

Before you start, make sure that you have installed Docker ([unknown ref software_reference/docker](#))

```
previous warning next (4 of 18) index
```

```
warning
```

```
I will ignore this because it is an external link.
```

```
> I do not know what is indicated by the link '#software_reference/docker'.
```

Location not known more precisely.

Created by function n/a in module n/a.

3) Install the Duckieton Shell

Install the Duckietown Shell using these instructions.

4) Fork the docs - *book* repo on the Github site

Fork one of the `docs - book` repos on the Github site (Figure 3.1).

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

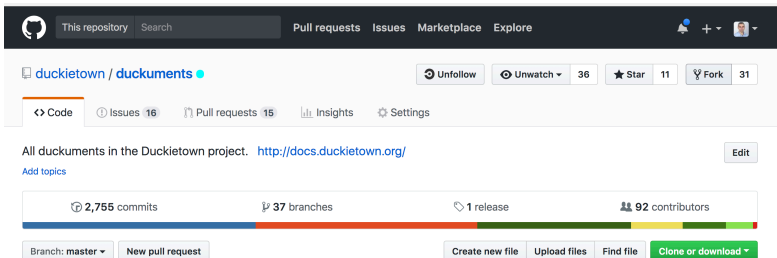


Figure 3.1

5) Checkout your fork locally

Check out the forked repository as you would do normally.

6) Initialize your folder

Go into the folder:

```
$ cd docs-book
```

When compiling a book for the first time, you need to run:

```
$ git submodule init
```

And:

```
$ git submodule update
```

7) Do your edits

Do your edits on your local copy. The source files are in the directory `book/ book`. The file and folder names start with numbers, these are used to determine the order that things show up in output.

8) Compile

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

```
$ dts docs build
```

Re-compile from scratch using:

```
$ dts docs clean  
$ dts docs build
```

If there are errors you should open `duckuments-dist/errors.html` and look at them and fix them.

9) Look at the local copy

Open the file `index.html` and navigate to whichever pages you have changed to make sure that they look the way you want them to.

10) Commit and push

Commit and push as you would do normally.

11) Make a pull request

Create a pull request to the original repository.

Option 1: Use the Github website:

Github offers a nice interface to create a pull request.

Option 2: Use the command-line program `hub`:

You can create a pull request from the command-line using hub ([unknown ref software_reference/hub](#))

```
previous warning next (5 of 18) index
```

```
warning
```

```
I will ignore this because it is an external link.
```

```
> I do not know what is indicated by the link '#software_reference/hub'.
```

```
Location not known more precisely.
```

```
Created by function n/a in module n/a.
```

```
:
```

```
$ hub pull-request
```

```
→ (unknown ref software\_reference/hub)
```

```
previous warning next (6 of 18) index
```

```
warning
```

```
I will ignore this because it is an external link.
```

```
> I do not know what is indicated by the link '#software_reference/hub'.
```

```
Location not known more precisely.
```

```
Created by function n/a in module n/a.
```

3.2. Using CircleCI

Circle CI makes it easier to check whether there are problems to be fixed.

1) Sign up on Circle

Sign up on the Circle CI service, at the link circleci.com.

2) Activate your build on Circle

Activate the building at the link:

```
https://circleci.com/setup-project/gh/ username /duckuments
```

where *username* is your Github username.

Click “start building”.

Make sure everything compiles on Circle:

Go to the URL:

```
https://circleci.com/gh/ username /duckuments
```

to see the status of your build.

You can also preview the results by clicking the “artifacts” tab and selecting `index.html` from the list.

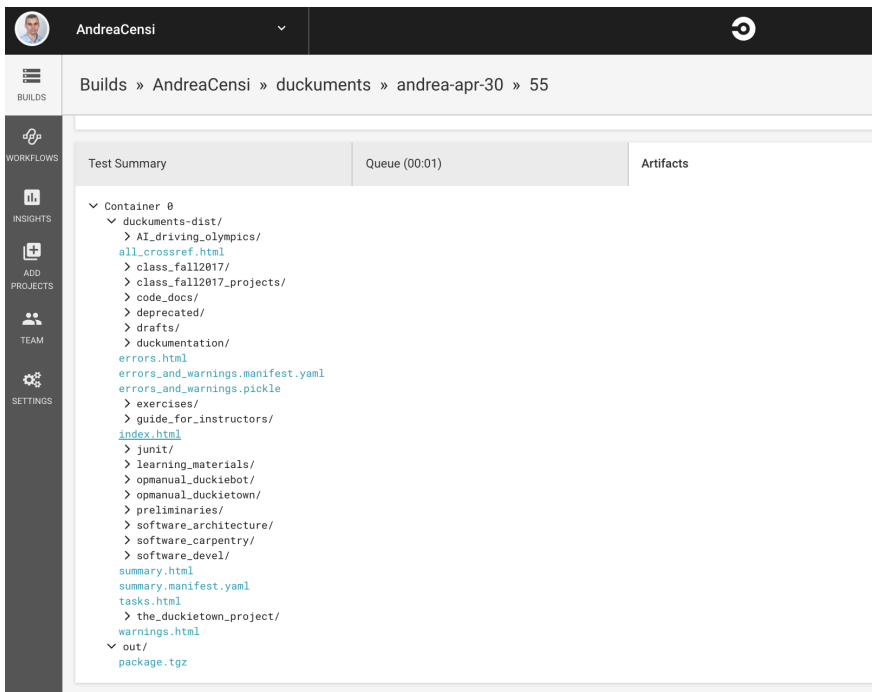


Figure 3.2

UNIT A-4

Troubleshooting for Duckumentation

4.1. Markduck problems

First, see the section Unit B-5 - Troubleshooting for common problems and their resolution.

Please report problems with the duckuments using the duckuments issue tracker.

Special notes:

- If you have a problem with a generated PDF, please attach the offending PDF.
- If you say something like “This happens for Figure 3”, then it is hard to know which figure you are referencing exactly, because numbering changes from commit to commit.

If you want to refer to specific parts of the text, please commit all your work on your branch, and obtain the name of the commit using the following commands:

```
$ git rev-parse HEAD
```

4.2. Problem: The building hangs

1) Cause: insufficient memory

This might be due to insufficient memory.

For example, on Mac, the default setting is 2GB of RAM. Try increasing it (Figure 4.1).

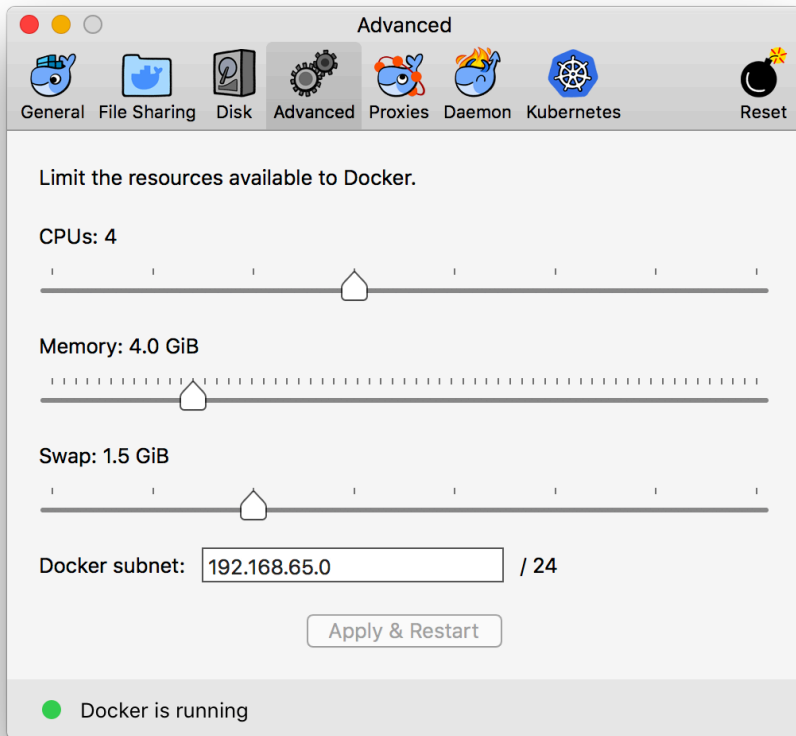


Figure 4.1

A workaround is to stop and restart the process (but without doing `docker system prune`).

2) Cause: downloading in the background

Sometimes the problem is that the code is downloading something in the background, for example a Git repository.

Currently this is hard to debug.

UNIT A-5

Markdown editors

We suggest the following:

- Find a visual Markdown editor that you like.
- Complement with a superb text-only editor.

5.1. Graphical Markdown editors

1) Typhora

Typhora is our favorite because it allows to preview LaTeX formulas.



5.2. Text editors good for Markdown

5.3. Atom

→ (unknown ref software_reference/atom)

previous warning next (7 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#software_reference/atom'.

Location not known more precisely.

Created by function n/a in module n/a.

5.4. PyCharm

→ (unknown ref software_reference/pycharm)

previous warning next (8 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#software_reference/pycharm'.

Location not known more precisely.

Created by function n/a in module n/a.

PART B

Markduck format

This part describes the Markdown dialect that is used in the documentation.

Contents

Unit B-1 - Basic Markduck guide	18
Unit B-2 - Special paragraphs	27
Unit B-3 - LaTeX constructs.....	32
Unit B-4 - Embedding videos.....	39
Unit B-5 - Troubleshooting.....	40
Unit B-6 - Knowledge Boxes.....	42

UNIT B-1

Basic Markduck guide

The Duckiebooks are written in Markduck, a Markdown dialect.

It supports many features that make it possible to create publication-worthy materials.

Contents

Section 1.1 - Markdown	18
Section 1.2 - Comments	18
Section 1.3 - Variables in command lines and command output	18
Section 1.4 - Character escapes	19
Section 1.5 - Keyboard keys	19
Section 1.6 - Figures.....	19
Section 1.7 - Subfigures	20
Section 1.8 - Shortcut for tables.....	21
Section 1.9 - Linking to documentation	22
Section 1.10 - Linking to other books	24

1.1. Markdown

The Duckiebook are written in a Markdown dialect.

- A tutorial on Markdown.

1.2. Comments

You can insert comments using the HTML syntax for comments: any text between “<!--” and “-->” is ignored.

```
# My section

<!-- this text is ignored -->

Let's start by...
```

1.3. Variables in command lines and command output

Use the syntax “![name]” for describing the variables in the code.

example

For example, to obtain:

```
$ ssh robot name.local
```

Use the following:

For example, to obtain:

```
$ ssh ![robot name].local
```

Make sure to quote (with 4 spaces) all command lines. Otherwise, the dollar symbol confuses the LaTeX interpreter.

1.4. Character escapes

Use the string “$” to write the dollar symbol “\$”, otherwise it gets confused with LaTeX math materials. Also notice that you should probably use “USD” to refer to U.S. dollars.

Other symbols to escape are shown in Table 1.1.

TABLE 1.1. SYMBOLS TO ESCAPE

use $	instead of \$
use `	instead of `
use <	instead of <
use >	instead of >

1.5. Keyboard keys

Use the `kbd` element for keystrokes.

example

For example, to obtain:

Press `a` then `Ctrl-C`.

use the following:

```
Press <kbd>a</kbd> then <kbd>Ctrl</kbd>-<kbd>C</kbd>.
```

1.6. Figures

To create a figure, use the element `figure`:

```
<figure>
  <figcaption>Hello</figcaption>
  
</figure>
```



Figure 1.1. Hello

1) More attributes

Use `class="caption-left"` to have the caption show up on the left rather than on the bottom:

```
<figure class="caption-left">
  <figcaption>Hello</figcaption>
  
</figure>
```

Figure 1.2. Hello



1.7. Subfigures

You can create subfigures by nesting `figure` elements:

```
<figure>
  <figcaption>Main caption</figcaption>
  <figure>
    <figcaption>Hello</figcaption>
    
  </figure>
  <figure>
    <figcaption>second</figcaption>
    
  </figure>
</figure>
```



(a) Hello



(b) second

Figure 1.3. Main caption

By default, they are displayed one per block. To make them flow horizontally, add `class="flow-subfigures"` to the external figure:

```
<figure class="flow-subfigures">
  <figcaption>Main caption</figcaption>
  <figure>
    <figcaption>Hello</figcaption>
    
  </figure>
  <figure>
    <figcaption>second</figcaption>
    
  </figure>
</figure>
```



(a) Hello



(b) second

Figure 1.4. Main caption

For any element, adding an attribute called `figure-id` with value `fig: figure ID` or `tab: table ID` will create a figure that wraps the element.

1.8. Shortcut for tables

The shortcuts `col2`, `col3`, `col4`, `col5` are expanded in tables with 2, 3, 4 or 5 columns.

The following code:

```
<col2 figure-id="tab:mytable" figure-caption="My table">
  <span>A</span>
  <span>B</span>
  <span>C</span>
  <span>D</span>
</col2>
```

gives the following result:

TABLE 1.2. MY TABLE

A	B
C	D

1) `labels-row1` and `labels-col1`

Use the classes `labels-row1` and `labels-col1` to make pretty tables like the following.

`labels-row1`: the first row is the headers.

`labels-col1`: the first column is the headers.

TABLE 1.3. USING CLASS="LABELS-COL1"

header A	B	C	1
header D	E	F	2
header G	H	I	3

TABLE 1.4. USING CLASS="LABELS-ROW1"

header A	header B	header C
D	E	F
G	H	I
1	2	3

1.9. Linking to documentation

1) Establishing names of headers

You give IDs to headers using the format:

```
### header title {#topic ID}
```

For example, for this subsection, we have used:

```
### Establishing names of headers {#establishing}
```

With this, we have given this header the ID “`establishing`”.

By convention, we use consistently dashes rather than underscores. For example, in-

stead of using the ID `#my_cool_section`, use `#my-cool-section`.

2) How to name IDs - and why it's not automated

Some time ago, if there was a section called

```
## My section
```

then it would be assigned the ID “my-section”.

This behavior has been removed, for several reasons.

One is that if you don't see the ID then you will be tempted to just change the name:

```
## My better section
```

and silently the ID will be changed to “my-better-section” and all the previous links will be invalidated.

The current behavior is to generate an ugly link like “autoid-209u31j”.

This will make it clear that you cannot link using the PURL if you don't assign an ID.

Also, I would like to clarify that all IDs are *global* (so it's easy to link stuff, without thinking about namespaces, etc.).

Therefore, please choose descriptive IDs, with at least two IDs.

E.g. if you make a section called

```
## Localization {#localization}
```

that's certainly a no-no, because “localization” is too generic.

✓

```
## Localization {#intro-localization}
```

Also note that you don't *need* to add IDs to everything, only the things that people could link to. (e.g. not subsections)

3) Linking from the documentation to the documentation

You can use the syntax:

```
[ ](# topic ID )
```

to refer to the header.

You can also use some slightly more complex syntax that also allows to link to only the name, only the number or both (Table 1.5).

TABLE 1.5. SYNTAX FOR REFERRING TO SECTIONS.

See `[](#establishing)`.

See Subsection 1.9.1 - Establishing names of headers

See ``.

See Establishing names of headers.

See ``.

See 1.9.1.

See ``.

See Subsection 1.9.1 - Establishing names of headers.

1.10. Linking to other books

It is possible to link from one book to another.

The syntax is a slightly extended syntax. Instead of using something like

See this `[interesting section](#interesting-section)`.

You need to use add `+other-book-id` before the `#`:

See this `[interesting section in another book](+other-book-id#interesting-section)`.

To find out what is the book ID, go to the current index of all the books, which, for `daffy`, is `https://docs.duckietown.org/daffy/`. Go to the book you want to link, and note the URL. The book ID is the part after `daffy`.

For example, this book is published at

`https://docs.duckietown.org/daffy/duckumentation/out/index.html`

The book ID is `duckumentation`.

To link to the entire book, use something like `+BOOKID#book`. (By convention, each book will have the main header assigned the header `#book:book`.)

Here is a list of some of the books as of August 2020:

[\(unknown ref opmanual_developer/book\)](#)

previous **warning** next (9 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#opmanual_developer/book'.

Location not known more precisely.

Created by function n/a in module n/a.

(unknown ref [opmanual_duckiebot/book](#))

previous **warning** next (10 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#opmanual_duckiebot/book'.

Location not known more precisely.

Created by function n/a in module n/a.

(unknown ref [opmanual_duckietown/book](#))

previous **warning** next (11 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#opmanual_duckietown/book'.

Location not known more precisely.

Created by function n/a in module n/a.

(unknown ref [opmanual_autolab/book](#))

previous **warning** next (12 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#opmanual_autolab/book'.

Location not known more precisely.

Created by function n/a in module n/a.

- (unknown ref [duckumentation/book](#))

previous **warning** next (13 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#documentation/book'.

Location not known more precisely.

Created by function `n/a` in module `n/a`.

- this book.

- (unknown ref `opmanual_sky/book`)

previous **warning** next (14 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#opmanual_sky/book'.

Location not known more precisely.

Created by function `n/a` in module `n/a`.

- (unknown ref `duckiesky_high_school_student/book`)

previous **warning** next (15 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#duckiesky_high_school_student/book'.

Location not known more precisely.

Created by function `n/a` in module `n/a`.

- (unknown ref `duckiesky_high_school/book`)

previous **warning** next (16 of 18) index

warning

I will ignore this because it is an external link.

> I do not know what is indicated by the link '#duckiesky_high_school/book'.

Location not known more precisely.

Created by function `n/a` in module `n/a`.

1) (For developers) How to update the list of books that can be crossreferenced.

The dictionary between book ID and URL is maintained in the repo `docs-build` in the file `books.crossref.yaml`. When a new book is added, the list needs to be updated and the image `duckietown/docs-build` be updated.

UNIT B-2

Special paragraphs and environments

2.1. Special paragraphs tags

The system supports parsing of some special paragraphs.

1) Special paragraphs must be separated by a line

A special paragraph is marked by a special prefix. The list of special prefixes is given in the next section.

There must be an empty line before a special paragraph; this is because in Markdown a paragraph starts only after an empty line.

This is checked automatically, and the compilation will abort if the mistake is found.

For example, this is invalid:

```
See: this book
See: this other book
```

This is correct:

```
See: this book

See: this other book
```

Similarly, this is invalid:

```
Author: author
Maintainer: maintainer
```

and this is correct:

```
Author: author

Maintainer: maintainer
```

2) Todos, task markers

```
TODO: todo    TOWRITE: towrite
```

To write: towrite

Task: task

Task: task

Assigned: assigned

| Assigned to: assigned

3) Notes and remarks

Remark: remark

| **Remark:** remark

Note: note

| **Note:** note

Warning: warning

Warning: warning

4) Troubleshooting

Symptom: symptom

| **Symptom:** symptom

Resolution: resolution

Resolution: resolution

5) Guidelines

Bad: bad

* bad

Better: better

✓ better

6) Questions and answers

Q: question

Q: question

A: answer

Answer: answer

7) Authors, maintainers, Point of Contact

Maintainer: maintainer

Maintainer: maintainer

Assigned: AndreaCensi

| Assigned to: AndreaCensi

8) References

See: see

→ see

Reference: reference

→ reference

Requires: requires

| **Requires:** requires

Results: results

Results: results

Next steps: next steps

Next: next steps

Recommended: recommended

Recommended: recommended

See also: see also

* see also

2.2. Other **div** environments

For these, note the rules:

- You must include `markdown="1"`.
- There must be an empty line after the first `div` and before the closing `/div`.

1) Example usage

```
<div class='example-usage' markdown="1">
This is how you can use `roscat`:
    $ roscat play log.bag
</div>
```

example

This is how you can use `roscat`:

```
$ roscat play log.bag
```

2) Check

```
<div class='check' markdown="1">
Check that you didn't forget anything.
</div>
```

Check before you continue

Check that you didn't forget anything.

3) Requirements

```
<div class='requirements' markdown="1">
```

```
List of requirements at the beginning of setup chapter.
```

```
</div>
```

KNOWLEDGE AND ACTIVITY GRAPH

List of requirements at the beginning of setup chapter.

UNIT B-3

Using LaTeX constructs in documentation

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Working knowledge of LaTeX.

Contents

Section 3.1 - Embedded LaTeX.....	32
o Proposition 1 - Proposition example	
o Listing 3.1 - Use of LaTeX code.	
o Table 3.1 - LaTeX environments and label prefixes	
o Definition 1 - My definition	
o Proposition 2 - My proposition	
o Remark 1 - My remark	
o Problem 1 - My problem	
o Example 1 - My example	
o Theorem 1 - My theorem	
o Lemma 1 - My lemma	
Section 3.2 - LaTeX equations.....	34
Section 3.3 - LaTeX symbols	35
Section 3.4 - Bibliography support	35
Section 3.5 - Embedding Latex in Figures through SVG.....	36
o Figure 3.1 - Image saved as svg	
o Figure 3.1a - Image saved as svg	
o Figure 3.1b - Image as PDF after processing	
Section 3.6 - Using the HTML equivalent of Latex environments.....	37
o Exercise 1 - Exercise title	
o Exercise 2 - Second exercise	
Section 3.7 - Alternative style	37
o Exercise 3 - Another exercise	
o Exercise 4 - Another exercise	

3.1. Embedded LaTeX

You can use \LaTeX math, environment, and references. For example, take a look at

$$x^2 = \int_0^t f(\tau) d\tau$$

or refer to Proposition 1 - Proposition example.

Proposition 1. (Proposition example) This is an example proposition: $2x = x + x$.

The above was written as in Listing 3.1.

```
You can use  $\LaTeX$  math, environment, and references.
For example, take a look at

\[
x^2 = \int_0^t f(\tau) \text{d}\tau
\]

or refer to [](#prop:example).

\begin{proposition}[Proposition example]\label{prop:example}
This is an example proposition:  $2x = x + x$ .
\end{proposition}
```

Listing 3.1. Use of LaTeX code.

For the LaTeX environments to work properly you *must* add a `\label` declaration inside. Moreover, the label must have a prefix that is adequate to the environment. For example, for a proposition, you must insert `\label{prop: name}` inside.

The following table shows the list of the LaTeX environments supported and the label prefix that they need.

TABLE 3.1. LATEX ENVIRONMENTS AND LABEL PREFIXES

definition	def: <i>name</i>
proposition	prop: <i>name</i>
remark	rem: <i>name</i>
problem	prob: <i>name</i>
theorem	thm: <i>name</i>
lemma	lem: <i>name</i>

Examples of all environments follow.

example

```
\begin{definition}[My definition] \label{def:lorem}
Lorem
\end{definition}
```

Definition 1. (My definition) Lorem

```
\begin{proposition}[My proposition] \label{prop:lorem}
Lorem
\end{proposition}
```

Proposition 2. (My proposition) Lorem

```
\begin{remark}[My remark]   \label{rem:lorem}
Lorem
\end{remark}
```

Remark 1. (My remark) Lorem

```
\begin{problem}[My problem]   \label{prob:lorem}
Lorem
\end{problem}
```

Problem 1. (My problem) Lorem

```
\begin{example}[My example]   \label{exa:lorem}
Lorem
\end{example}
```

Example 1. (My example) Lorem

```
\begin{theorem}[My theorem]   \label{thm:lorem}
Lorem
\end{theorem}
```

Theorem 1. (My theorem) Lorem

```
\begin{lemma}[My lemma]   \label{lem:lorem}
Lorem
\end{lemma}
```

Lemma 1. (My lemma) Lorem

```
I can also refer to all of them:
[](#def:lorem),
[](#prop:lorem),
[](#rem:lorem),
[](#prob:lorem),
[](#exa:lorem),
[](#thm:lorem),
[](#lem:lorem).
```

I can also refer to all of them: Definition 1 - My definition, Proposition 2 - My proposition, Remark 1 - My remark, Problem 1 - My problem, Example 1 - My example, Theorem 1 - My theorem, Lemma 1 - My lemma.

3.2. LaTeX equations

We can refer to equations, such as **(1)**:

$$2a = a + a$$

This uses `align` and contains (2) and (???)

$$\begin{aligned} a &= b \\ &= c \end{aligned}$$

We can refer to equations, such as `\eqref{eq:one}`:

```
\begin{equation}
  2a = a + a          \label{eq:one}
\end{equation}
```

This uses `\align`` and contains `\eqref{eq:two}` and `\eqref{eq:three}`.

```
\begin{align}
  a &= b          \label{eq:two} \\
  &= c          \label{eq:three}
\end{align}
```

Note that referring to the equations is done using the syntax `\eqref{eq: name}`, rather than `[#eq: name]`.

3.3. LaTeX symbols

You can place any LaTeX symbols definition in files called `*.symbols.tex`.

These will be included as preamble.

For example, this repository contains a file `a.symbols.tex` containing:

```
\newcommand{\mysymbol}{\text{This is defined in a.symbols.tex}}
```

So then when we create an equation with:

```
$$ \mysymbol $$
```

It gets rendered as:

This is defined in a.symbols.tex

3.4. Bibliography support

You need to have installed `bibtex2html`.

The system supports Bibtex files.

Place `*.bib` files anywhere in the directory.

Then you can refer to them using the syntax:

```
[ ](#bib: bibtex ID )
```

For example:

```
Please see [ ](#bib:siciliano07handbook).
```

Will result in:

Please see [1].

Somewhere in the document, add the following:

```
&lt;div id="put-bibliography-here"&gt;&lt;/div&gt;
```

3.5. Embedding Latex in Figures through SVG

KNOWLEDGE AND ACTIVITY GRAPH

Requires: In order to compile the figures into PDFs you need to have Inkscape installed. Instructions to download and install Inkscape are [here](#).

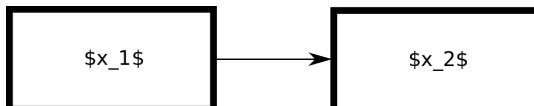
To embed latex in your figures, you can add it directly to a file and save it as [file-name](#).svg file and save in the /docs directory in a subfolder called assets/svg2pdf.

You can then run:

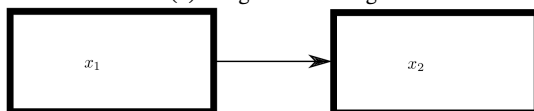
```
$ make process-svg-figs
```

And the SVG file will be compiled into a PDF figure with the LaTeX commands properly interpreted.

You can then include the PDF file in a normal way (Section 1.6 - Figures) using [file-name](#).pdf as the filename in the `` tag.



(a) Image saved as svg



(b) Image as PDF after processing

Figure 3.1. Embedding LaTeX in images

It can take a bit of work to get the positioning of the code to appear properly on the figure.

3.6. Using the HTML equivalent of Latex environments

Contents

Exercise 1 - Exercise title

Exercise 2 - Second exercise

You can create an exercise as follows:

```
<div id="exercise:my-exercise" class="exercise" title="Exercise title">
  This is an exercise labeled "exercise:my-exercise".
</div>

<div class="exercise" title="Second exercise">
  This is an exercise not labeled.
</div>
```

Referring to the exercise: [](#exercise:my-exercise) or [](#my-exercise).

| **Exercise 1.** This is an exercise labeled "exercise:my-exercise".

| **Exercise 2.** This is an exercise not labeled.

Referring to the exercise: Exercise 1 - Exercise title or Exercise 1 - Exercise title.

3.7. Alternative style

If you have block content element (e.g. code blocks), you must use headers as html cannot contain block-level Markdown.

You must use something like the following. The marker `end` tells the system to stop the level 4 section.

```
#### Another exercise {#exercise:another}

This is another exercise with block content:

    $ echo hello

<end/>

This is part of the level 2 section.

Another exercise starts for the rest of the section.

#### Another exercise {#exercise:second}

contents

<end/>

After the second exercise.
```

Exercise 3. Another exercise.

This is another exercise with block content:

```
$ echo hello
```

This is part of the level 2 section.

Another exercise starts for the rest of the section.

Exercise 4. Another exercise.

contents

After the second exercise.

UNIT B-4

Embedding videos

It is possible to embed Vimeo videos in the documentation.

Note: Do not upload the videos to your personal Vimeo account; they must all be posted to the Duckietown Engineering account.

Note: The videos must be *public* for the embedding to work. If the video is not public the process will fail mentioning the video is not found (true, because it is not public).

This is the syntax:

```
<dtvideo src="vimeo:vimeo ID"/>
```

example

For example, this code:

```
<figure id="example-embed">  
  <figcaption>Cool Duckietown by night</figcaption>  
  <dtvideo src="vimeo:152825632"/>  
</figure>
```

produces this result:



Figure 4.1. Cool Duckietown by night

Depending on the output media, the result will change:

- On the online book, the result is that a video player is embedded.
- On the e-book version, the result is that a thumbnail is produced, with a link to the video;
- (future improvement) On the dead-tree version, a thumbnail is produced with a QR code linking to the video.

UNIT B-5

Troubleshooting

Contents

Section 5.1 - Troubleshooting errors in the compilation process.....	40
Section 5.2 - Not properly starting a list	40
Section 5.3 - Forgetting <code>markdown=1</code>	41
Section 5.4 - Code inside html	41

5.1. Troubleshooting errors in the compilation process

Symptom: “Invalid XML”

Resolution: “Markdown” doesn’t mean that you can put anything in a file. Except for the code blocks, it must be valid XML. For example, if you use “>” and “<” without quoting, it will likely cause a compile error.

Symptom: “Tabs are evil”

Resolution: Do not use tab characters. The error message in this case is quite helpful in telling you exactly where the tabs are.

Symptom: The error message contains `ValueError: Suspicious math fragment 'KEYMATHS000ENDKEY'`

Resolution: You probably have forgotten to indent a command line by at least 4 spaces. The dollar in the command line is now being confused for a math formula.

5.2. Not properly starting a list

There must be an empty line before the list starts.

This is correct:

```
I want to learn:
- robotics
- computer vision
- underwater basket weaving
```

This is incorrect:

```
I want to learn:
- robotics
- computer vision
- underwater basket weaving
```

and it will be rendered as follows:

I want to learn: - robotics - computer vision - underwater basket weaving

5.3. Forgetting `markdown=1`

If you forget `markdown=1` the contents of the tag will not be processed.

For example, this:

```
<div class="exercise" title="My exercise title">
```

```
    Open a new terminal and navigate to `~/ssh` and open the file named  
    `config`.
```

```
</div>
```

results in:

Exercise 5. Open a new terminal and navigate to `~/ssh` and open the file named `config`.

5.4. Code inside html

Exercise 6. Here is a code block: `$ code block` Another block: `pythonomtyrngimp` or `tAny` After block: `code block`

UNIT B-6

The Knowledge and Activity Graph Environment

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Linking between pages of the duckumentation

Recommended: Markduck, special paragraphs

Results: Knowledge how to write a knowledge box

6.1. Basics

Every page, on every book, should have a ‘Knowledge Box’ - also called ‘Knowledge and Activity Graph’. It’s the blue box you see above. In markdown, a knowledge box looks like this:

```
<div class='requirements' markdown="1">
Requires: Important prerequisite XY
Requires: ...
Results: Infinite wisdom
</div>
```

6.2. Purpose

Knowledge boxes give structure to the duckumentation. How do you know where to start? For example by following back links in knowledge boxes! Or by having an automated system follow back those references for you.

Also, imagine if dead references or circular dependencies were detected automatically. Authors could fix this right away, and you would have an easier time navigating the duckumentation. So: Add a knowledge box!

6.3. Guidelines

Here are dos and don’ts how to write useful knowledge boxes:

✓ **One knowledge box per page.** (*Might change in the future.*)

Organize your pages in a way that one knowledge box per page makes sense. Optimally, one page should be about one distinct topic, or describe how to create one distinct thing. You can use most of the existing duckumentation as reference for a good “topic granularity”.

- ✓ Use ‘Requires:’, ‘Recommended:’ and ‘Results:’ markers.

Start each line in a knowledge box with one of them.

- ✓ In “Requires:” lines, link to pages that provide the requirement whenever possible.
- ✓ Link only to CSS-identifiers of other pages, not specific sections on that page. (This might change in the future.)
- ✓ One requirement or result per line.

- ✗ A list of multiple requirements in one line.

Do not: ([unknown ref opmanual_duckiebot/setup-duckiebot](#))

```
previous warning next (17 of 18) index
```

```
warning
```

```
I will ignore this because it is an external link.
```

```
> I do not know what is indicated by the link '#opman-
ual_duckiebot/setup-duckiebot'.
```

```
Location not known more precisely.
```

```
Created by function n/a in module n/a.
```

- ✓ Use short descriptive texts, with good keywords that identify the requirement or result.

- ✓ **Optimal:** Use the same description as in other knowledge boxes, when referring to the same entity. (Exception: If the other page’s description violates these guidelines.)

- ✗ Long texts, with many unrelated keywords.

Do not:

```
Results: A correctly configured Duckiebot SD card in configuration DB18.
After assembling the Duckiebot, this will allow you to start it, con-
nect to the internet, and get going.
```

```
Requires: Software-X, with all requirements for the ROS example
```

- ✓ Put all text that does not directly describe the requirement between round brackets
- ** Omit unnecessary words.

Do:

Results: A configured Duckiebot SD card in configuration DB18. (After assembling the Duckiebot, this will allow you to start it, connect to the internet, and get going.)

Requires: Software-X (with all requirements for the ROS example)

✓ For estimated time needed, use a ‘Requires:’ marker.

For example:

Requires: Approximately 1 hour 30 minutes.

(“1h30”, or “1h, 30 min” works as well.)

✓ If you want, you can use the instructions [here](#)(Pointer to beta/draft material that was removed - instructions-refer-to-data)

previous warning (18 of 18) index

warning

This link points to beta/draft material removed

Location not known more precisely.

Created by function n/a in module n/a.

to pick a name of an entity that the EOD system knows already, from the data files, or add a new entity there via pull request.

** This *might* change in the future.

PART C
Documentation style guide

This part describes the style guide for our documentation.

Contents

Unit C-1 - Style guide	46
-------------------------------------	-----------

UNIT C-1

Style guide

This chapter describes the conventions for writing the technical documentation.

1.1. Organization

The documentation is divided into books, parts (labeled ‘part:’) and units (with no CSS prefix).

To create a new part, put `{#part:name status=STATUS}` after the header, like so:

```
## Safety {#part:safety status=ready}
```

1.2. General guidelines for technical writing

The following holds for all technical writing.

- The documentation is written in correct English.
- Do not say “should” when you mean “must”. “Must” and “should” have precise meanings and they are not interchangeable. These meanings are explained in this document.
- “Please” is unnecessary in technical documentation.
 - ✘ “Please remove the SD card.”
 - ✓ “Remove the SD card”.
- Do not use colloquialisms or abbreviations.
 - Bad: “The pwd is ubuntu.”
 - Better: “The password is ubuntu.”
 - ✘ “To create a ROS pkg...”
 - ✓ “To create a ROS package...”
- Python is capitalized when used as a name.
 - ✘ “If you are using python...”
 - ✓ “If you are using Python...”
- Do not use emojis.
- Do not use ALL CAPS.
- Make infrequent use of **bold statements**.
- Do not use exclamation points.

1.3. Style guide for the Duckietown documentation

- The English version of the documentation is written in American English. Please

note that your spell checker might be set to British English.

- ✗ **behaviour**
 - ✓ **behavior**
 - ✗ **localisation**
 - ✓ **localization**
- It's ok to use “it's” instead of “it is”, “can't” instead of “cannot”, etc.
 - All the filenames and commands must be enclosed in code blocks using Markdown backticks.
 - ✗ “Edit the `~/ssh/config` file using `vi`.”
 - ✓ “Edit the `~/ssh/config` file using `vi`.”
 - `Ctrl-C`, `ssh` etc. are not verbs.
 - ✗ “`Ctrl-C` from the command line”.
 - ✓ “Use `Ctrl-C` from the command line”.
 - Subtle humor and puns about duckies are encouraged.


1.4. Writing command lines

Use either “`laptop`” or “`duckiebot`” (not capitalized, as a hostname) as the prefix for the command line.

For example, for a command that is supposed to run on the laptop, use:

```
laptop $ cd ~/duckietown
```


It will become:

```
 $ cd ~/duckietown
```

For a command that must run on the Duckiebot, use:

```
duckiebot $ cd ~/duckietown
```

It will become:

```
 $ cd ~/duckietown
```

If the command is supposed to be run on both, omit the hostname:

```
$ cd ~/duckietown
```

Other rules:

- For a container use `container`.
- For a container on a Duckiebot use `duckiebot-container`.

- For a container on the laptop use `laptop-container`.

This:

```
container $ command
```

will become:



```
$ command
```

This:

```
duckiebot-container $ command
```

will become:



```
$ command
```

This:

```
laptop-container $ command
```

will become:



```
$ command
```

1.5. Frequently misspelled words

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

1.6. 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.

(If people need to be told how to edit a file, Duckietown is too advanced for them.)

1.7. 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.

1) Troubleshooting

| Symptom: This strange thing happens.

Resolution: Maybe the camera is not inserted correctly. Remove and reconnect.

| Symptom: This other strange thing happens.

Resolution: Maybe the plumbus is not working correctly. Try reformatting the plumbus.

PART D

Slides

Contents

Unit D-1 - Creating slides	51
? Making slides.....	52

UNIT D-1

Creating slides

See this presentation: ? Making slides

Making slides

To create slides, use `type=slides` for an H1 header:

```
# Making slides {#making-slides type=slides status=ready nonumber=1}
```

To create slides, use the attribute ``type=slides`` for an H1 header:

1.1. Next slides

Use second-level headers to make subsequent slides:

```
# Making slides {#making-slides type=slides status=ready}
```

To create slides, use ``type=slides`` for an H1 header:

...

```
## Next slides
```

Use second-level headers to make subsequent slides:

1.2. Stepping

Use the symbol  to make the corresponding fragment appear on click.

```
* Step 1 ►
* Step 2 ►
* Step 3 ►
```

- Step 1 ►
- Step 2 ►
- Step 3 ►

1.3. Maths

Latex still works here.

A simple test for math:

$$a + b \geq \sqrt{c}$$

1.4. Stepping through equations

You can also step through equations:

Consider: ▶

\$\$

$a = b$ ▶

\$\$

Then we get: ▶

\$\$

$c = d$ ▶

\$\$

Consider: ▶▶

$a = b$ ▶

Then we get: ▶▶

$c = d$ ▶

Stepping through partial parts of equations is not supported.

1.5. Sub-slides

You can have “sub slides” to make the presentation nonlinear.

Defining subslices: Create a subslice by using header `h3`:

```
### Subslide 1
```

This is lower.

```
### Subslide 2
```

This is even lower.

Showing subslices: Press down to show the subslices.

Showing the slides map: Press `(ESC)` again to look at the slides map.

1) Subslide 1

This is lower.

2) Subslide 2

This is even lower.

1.6. Presentation mode

Press the key `(S)` to enter presenters mode.

Press `ESC` to exit presenter mode.

1.7. Presenter notes

Use a blockquote at the end of a slide to encode the presenter notes.

```
## Presenter notes
```

Use a blockquote at the end of a slide to encode the presenter notes:

```
> These are presenter notes that will appear in presenter mode.
```

These are presenter notes that will appear in presenter mode.

1.8. Under the hood

- All of this is built on top of `reveal.js`.
- Please see `reveal.js` for the complete list of features.

1.9. Figures

All other duckuments features work as expected.

Example of a figure:



Figure 1.1

```
<figure class="stretch">  
    
</figure>
```

1.10. Subfigures

Subfigures with animation:

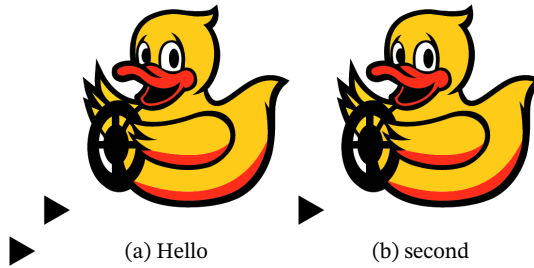


Figure 1.2. Main caption

```

<figure class="flow-subfigures"> ▶
  <figcaption>Main caption</figcaption>
  <figure> ▶
    <figcaption>Hello</figcaption>
    
  </figure>
  <figure> ▶
    <figcaption>second</figcaption>
    
  </figure>
</figure>

```

1.11. Cross references

You can link to chapters and vice-versa: [Unit D-1 - Creating slides](#)

You can link to chapters and vice-versa: `[](#creating-slides)`

[Link to the previous slide.](#)

1.12. References

[1] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.