



# hiro

## Editorial Guide for Arago Documentation

Markus Napp

Version 1.4 DRAFT, 17/08/2017

# Table of Contents

<u>1. Contributing to HIRO Documentation</u>	1
1.1. You will need	1
1.2. How to install	1
1.3. How to contribute	2
<u>2. File Format and Workflow</u>	10
2.1. File extensions	10
2.2. Diagrams	11
2.3. Advanced features	13
<u>3. Document Structure</u>	14
3.1. Writing style	14
3.2. Document header	14
3.3. Sections	14
3.4. Paragraphs	14
3.5. Inline Anchors	15
3.6. Cross References	15
3.7. Images in tables	15
3.8. Code in tables	15
3.9. Included files	15
3.10. Ifdef and Ifndef Statements	16
3.11. Restrictions and Tips	16
<u>4. Formatting Instructions</u>	18
4.1. Text formatting	18
4.2. Special text formatting	18
4.3. Underline text	18
4.4. Strike through	18
4.5. Special Characters	19
4.6. Warning boxes (Admonition)	19
4.7. File names	20
4.8. Package names	21
4.9. Path names	21
4.10. Ports	21
4.11. Factory Names	21
4.12. Class Names	22
4.13. Function calls	22
4.14. URLs	22
4.15. Variables/Attributes	23
4.16. Images	24
4.17. Code	25
4.18. Lists	25

4.19. Tables .....	27
<u>5. Document Attributes</u> .....	29
5.1. Header Attributes .....	29
5.2. Freetext attributes .....	29
<u>6. Export to PDF</u> .....	31
6.1. Install toolchain .....	31
6.2. Export a document .....	31
6.3. Correct linking for PDF .....	32
6.4. Images in PDF .....	33
<u>7. Maintaining the HIRO Documentation Repository</u> .....	34
7.1. AsciiBinder .....	34
7.2. GitHub Best Practices .....	36

# Chapter 1. Contributing to HIRO Documentation



We highly value contribution. Please join the [hiro-documentation-collaborators](#).

## 1.1. You will need

- [git](#)
- A [GitHub account](#)
- Membership in the [hiro-documentation-collaborators](#) team, you can request that on GitHub
- [Atom](#) editor (or your favorite text editor)
  - [Atom Flight Manual](#)



If you're using Atom install the packages "language-asciidoc" and "asciidoc-preview" by running

```
apm install language-asciidoc asciidoc-preview
```

or through the "Packages" dialog in Atom.

- [Asciidoctor.js Live Preview Extension](#) for [Chrome](#) or [Firefox](#)



For the Chrome extension you need to enable "Allow access to file URLs" in "chrome://extensions/"

- [Sketch](#) (with the default settings) to create arrows and annotations for screenshots.

## 1.2. How to install



This is under construction

- `gem install rouge`

### 1.2.1. Windows

- [Install Ruby](#)

- [Install AsciiDoctor PDF](#)

```
gem install --pre asciidoctor_pdf
```

- [Atom Editor](#)

### 1.2.2. Mac OS

- [Brew package manager](#)
- [brew install ruby](#)
- [Install AsciiDoctor PDF](#)

```
gem install --pre asciidoctor_pdf
```

- [Atom Editor](#)

### 1.2.3. Linux

- [Install AsciiDoctor PDF](#)

```
gem install --pre asciidoctor_pdf
```

- [Atom Editor](#)

## 1.3. How to contribute

The short form is:

- Clone repository
- Branch off
- Make changes
- Commit changes
- Push branch
- Make pull request
- Rework

### 1.3.1. Clone the repository

Once you have set up git on your machine, created an account on GitHub and have access to the collaborators team, you can start working.

Navigate to your favorite directory and run:

```
git clone https://github.com/arago/hiro-documentation
```

You will be asked for credentials. Enter your GitHub username and password. This will create a called "hiro-documentation" on your machine that contains the full state of the GitHub repository; including all sub branches.

### 1.3.2. Checkout the 'right' branch

Most of the work that is going into the documentation is done on the 'develop' branch. If you do not know which version your documentation will pertain to, assume that it is for the latest version. If you have a contribution that will affect multiple versions, please contact the maintainer or create multiple pull requests (we will cover that later).

For this guide we assume you want to add some content to the latest version so you must checkout "develop".

```
git checkout develop
```

Git will load the develop branch into the directory.



You can check which branch you are on by using `git branch`

### 1.3.3. Make some changes

Now you can use your editor of choice to make some changes to the documentation. You can create, delete, modify any file you want. It will not affect the online repository because you are working on your offline copy.

The syntax used is **AsciiDoc**. If you have installed one of the browser extensions you can preview your ".adoc" files like they would look in AsciiDoctor. They will look slightly different in the final documentation template but it should be sufficient to account for correct syntax.



AsciiDoctor is way more powerful and has quite a large **User Manual**



Due to the nature of the used toolchain, there are some counterintuitive behaviors of included pages, links and images. The maintainer will work with you through the Request/Review process to create fully functional documents.

### 1.3.4. Branch off your changes



You can also branch off before making changes but follow this beginners guide for now.



It's important that you have checked out the branch that you actually want to branch off into your own branch.

Once you have made enough changes, you should create your own branch so they can be cleanly merged into "develop" by the maintainers. Run:

```
git checkout -b <type-your-new-branch-here> ①
```

① For example "git checkout -b my-fixes"

This now creates a local branch called **my-fixes** of the repository that contains your changes.



Give your branch a name that allows identifying it either by your name/handle or the type of change you have made or even a combination of both (e.g. **mnapp-cluster-docs**).

### 1.3.5. Add and Commit your changes

In the next step you must "commit" your changes to the branch. This means you select changes that will later be tracked under the same commit number. You can do this very selectively or with a lot of changes at once.

To find out which changes are not yet committed run:

```
git status
```

You should see a list of red paths and filenames that correspond with the changes you have made earlier. Now you will select which files (read: which changes) are going to be added to your next commit.



If file and pathnames are green you have already added them to your next commit. To remove added changes run **git reset** this will reset the branch to the last committed change.

We assume here that you simply wish to add all your current changes to the branch so you simply



run:

```
git add . ①
```

① Add the current directory (.)

You will see the same files and paths now show up in green.

Now it's time to create a **commit** that contains your added changes.

```
git commit -m "<Type a useful message here>" ①
```

① All commits must have a commit message that can tell the maintainer and other developers what you are trying to achieve with the change.

The correct format for a commit message is:

- "Fix spelling in installation guide"
- or
- "Add new page for administration of component X"



Imagine your commit message as a spoken word "patch" and describe what your change will do once applied.

You will see something like this:

```
git commit -m "Add contributing info first version"
[my-fixes 7fca232] Add contributing info first version
2 files changed, 141 insertions(+)
create mode 100644 CONTRIBUTING.adoc
create mode 100644 create_pr.png
```

This means your commit has the shortened ID "7fca232".

### 1.3.6. Push your changes

Now you have a local branch with some changes that do not exist in the online repository. To get the changes integrated you need to first upload your branch to the central repository and then request adding the changes.





The activity of combining changes from two repositories/branches is called a "merge" and what you want to do here is a "Pull request" (read: Pull your changes into the central repository) so the maintainer can merge them.

You will run:

```
git push origin my-fixes ①
```

- ① You will *push* your newly created branch **my-fixes** to the remote repository "origin" (by default this is the repository that you cloned in the first step).

You will see something like:

```
git push origin my-fixes
Counting objects: 134, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (77/77), done.
Writing objects: 100% (134/134), 35.16 KiB | 0 bytes/s, done.
Total 134 (delta 95), reused 96 (delta 57)
remote: Resolving deltas: 100% (95/95), completed with 34 local objects.
To https://github.com/arago/hiro-documentation
* [new branch]      my-fixes -> my-fixes
```

This will create a new branch on GitHub called **my-fixes** which contains all of your commits.



Only the commits that you have pushed from your local repository will show up there. Any changes that you have not committed will not be tracked in the remote branch.

### 1.3.7. Create a pull request

Now you navigate to the **branches list** of the project on GitHub.

Locate your branch and click on "New Pull Request".

arago / hiro-documentation Private
Unwatch 26 Unstar 2 Fork 5

Code Pull requests 0 Projects 0 Pulse Graphs Settings

Overview Yours Active Stale All branches Search branches...

**Default branch**

develop Updated 2 hours ago by fotto
Default
Change default branch

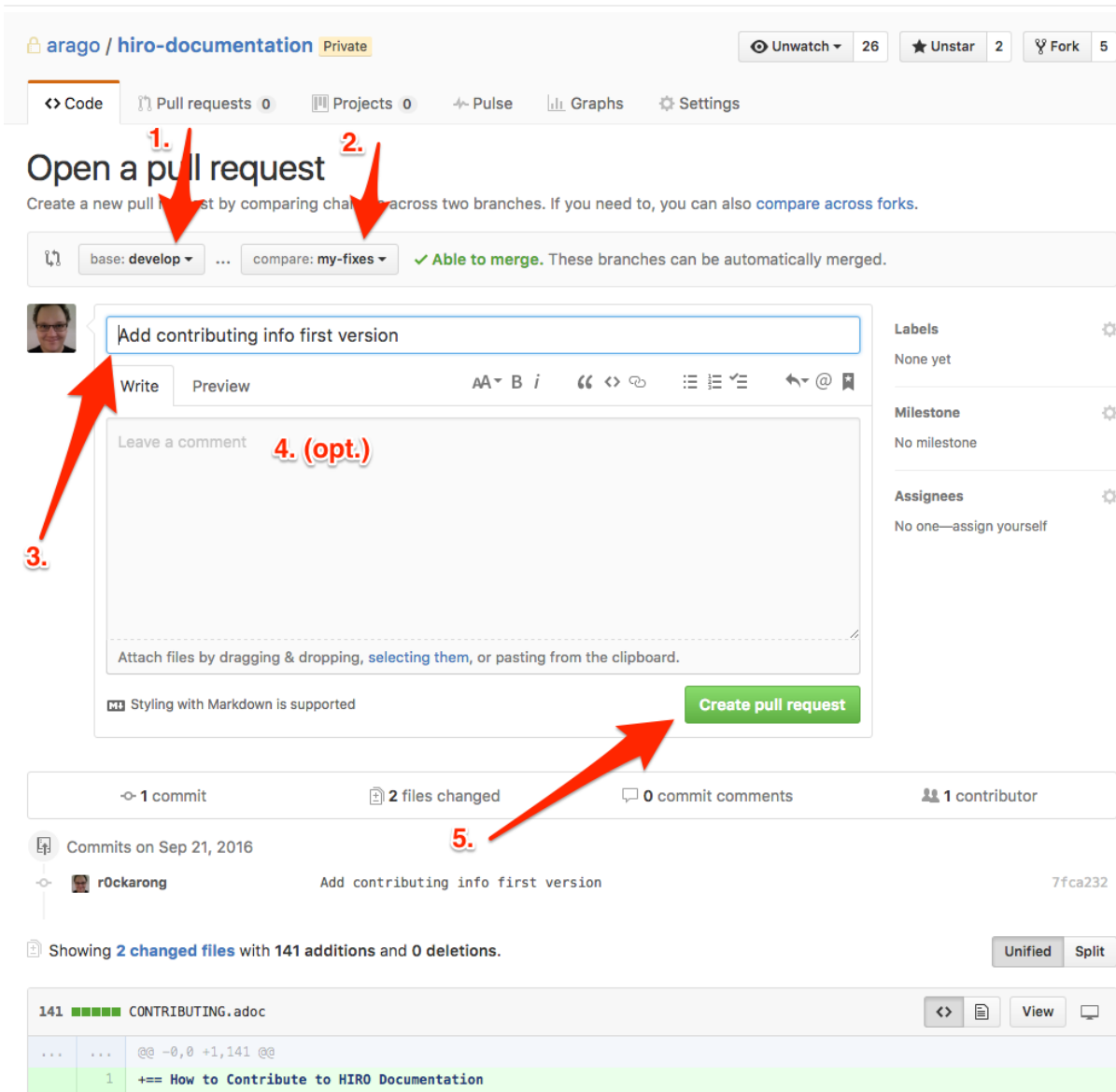
**Your branches**

my-fixes Updated 19 hours ago by r0ckarong	1   0	New pull request	
5.4.0 Updated 20 hours ago by r0ckarong	27   106	New pull request	
master Updated 23 days ago by r0ckarong	613   1	New pull request	

**Active branches**

my-fixes Updated 19 hours ago by r0ckarong	1   0	New pull request	
5.4.0 Updated 20 hours ago by r0ckarong	27   106	New pull request	
bmc-cmbd-connect-inst Updated 5 days ago by Daniel Polanowski	30   1	New pull request	
master Updated 23 days ago by r0ckarong	613   8	New pull request	

In the next window you select



arago / hiro-documentation Private

Unwatch 26 Unstar 2 Fork 5

Code Pull requests 0 Projects 0 Pulse Graphs Settings

## 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: `develop` ... compare: `my-fixes` ✓ Able to merge. These branches can be automatically merged.

Add contributing info first version

Write Preview

Leave a comment 4. (opt.)

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

Styling with Markdown is supported

Create pull request

1 commit 2 files changed 0 commit comments 1 contributor

Commits on Sep 21, 2016

r0ckarong Add contributing info first version 7fca232

Showing 2 changed files with 141 additions and 0 deletions.

Unified Split

141 CONTRIBUTING.adoc

```
@@ -0,0 +1,141 @@
1 +== How to Contribute to HIRO Documentation
```

1. The base branch `base:develop`
2. Which branch it will be compared to `compare:my-fixes`
3. The commit message of your last change will become the overall description, if your commit message is not great please add a good description here
4. (Optional) Add a detailed description of what you have changed, this might be necessary if the maintainer does not directly have insight in why you might have changed certain structures or content.
5. Submit your pull request



Just creating the pull request does not change anything in the main repository. The maintainer must manually agree to take in the changes. This is the place where all review and content discussion should take place after the initial work is done.

### 1.3.8. Work with the maintainer

Now your pull request will be reviewed and if everything went smoothly it will be merged straight away. You might receive comments or questions in your pull request about the changes so please keep an eye out for those.



Please consider that your contribution must be of good (enough) quality to be accepted. The maintainer(s) will not merge pull requests that must be reworked significantly. During review you might be asked to correct formatting and style to live up to the standards of the documentation. The editorial guide is your weapon of choice to develop good style.

If certain topics are not (sufficiently covered) in the editorial guide please add them or ask the maintainer to clarify for all future contributors.

Once your changes have been merged the remote branch will be deleted. Don't worry you can push your local branch again at any time.

If you are a maintainer for the docs please check out the [Information for maintainers](#)



Thanks for contributing to the HIRO documentation!

# Chapter 2. File Format and Workflow

- [AsciiDoc Quick Syntax Reference](#)
- [AsciiDoctor User Manual - Long Read](#)
- [AsciiDoc Writers Guide](#)
- [AsciiDoc Best Practices](#) (Need to review)
- [Alternate AsciiDoc Manual](#)
- [Great AsciiDoc Resource](#)
- [AsciiBinder Users Guide](#)
- [AsciiBinder Maintainers Guide](#)

## 2.1. File extensions

It's much preferred to use the suggested default file types and extensions:

### 2.1.1. Documents

Text documents that are part of the contribution should be in AsciiDoctor syntax ([.adoc](#)). This allows us to efficiently manage changes and ensure a corporate identity and style across the published documents. If you need to generate a "hard" version of a specific document version you can [export it to PDF](#) using the `asciidoctor-pdf` tool.

Exported PDFs **must** be placed in the [\\_pdfs](#) of the root project. The `gitignore` file is set up to ignore them everywhere else and will get omit them from commits and branches.

### 2.1.2. Images

Preferred image formats are:

- Portable Network Graphics - PNG ([.png](#))
- Joint Photographic Experts Group - JPEG ([.jpg/.jpeg](#))
- Scalable Vector Graphics - SVG ([.svg](#))



- PNG is preferred for screenshots
- SVG for exported diagrams

Images need to be placed in a called [images](#) in the same directory as the [.adoc](#) file you wish to use

the picture in.

### 2.1.3. Diagrams

If you store Ascii type diagrams in separate files please use their respective type as the file extension, e.g.:

- DITAA: .ditaa
- Mermaid: .mermaid
- PlantUML: .plantuml
- Shaape: .shaape

This makes filtering for this type of files much more convenient. These "extensions" can be used identically to regular file extensions in the include macro.



You can try out various diagram formats with this helpful [online tool](#) TIP: To create basic DITAA diagrams you can use the [AsciiFlow Online editor](#)

## 2.2. Diagrams

You can include various Ascii art type diagrams (plantUML, ditaa etc.) to your documents. To do so either put them as inline blocks into the document or include a block macro to your page.

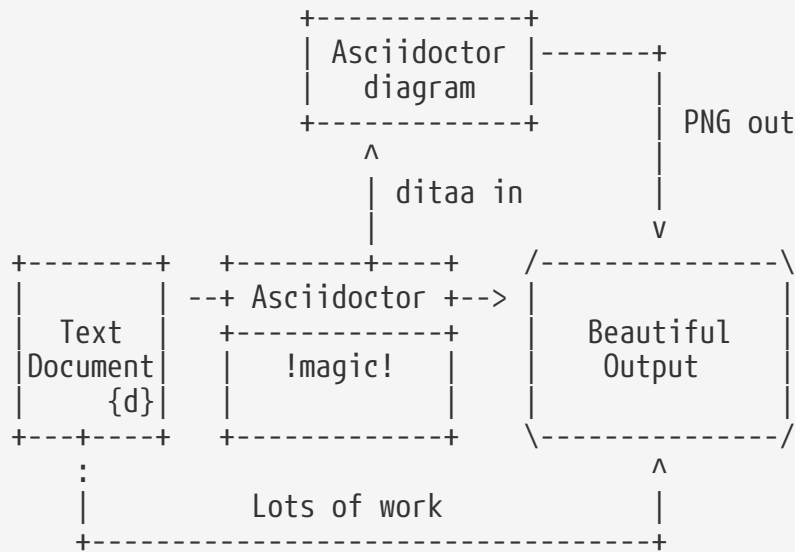


The block name/macro is determined by the format of the diagram you wish to include. Please refer to the [list of supported diagram types](#) to find out what you need to use.

For example to add a ditaa diagram, you add the diagram inline:

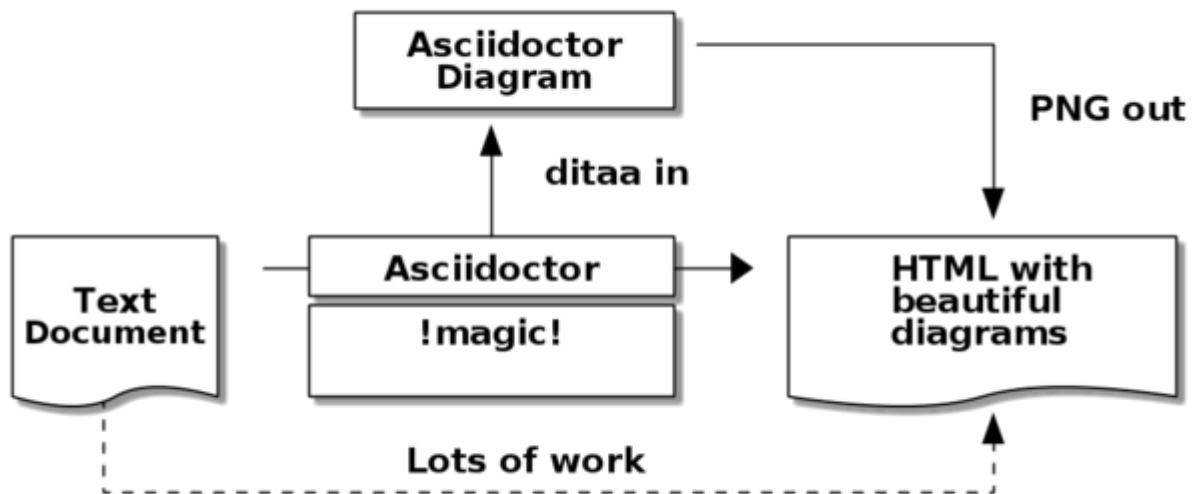
[ditaa]

....



....

This is processed by the [graphviz library](#) and other components to be displayed like this:



To find out which formats are supported check out the documentation for [Asciidoctor-diagram](#).

### 2.2.1. External diagrams

You can also store diagrams in separate files. To include such a file you must include via a macro for example the above diagram stored in a [ditaa.txt](#):



```
ditaa::path/to/my.file[My diagram]
```



You must use an absolute path from the project root to link to the diagram otherwise AsciiBINDER will not find it.

## 2.3. Advanced features



There are some **advanced options** for DITAA.

The Dita extension supports the options **scale**, **antialias**, **separation**, **round-corners**, **shadows** and **debug**

Options can be specified per block as `<option>=<value>` or globally as a document attribute of the form `:ditaa-option-<option>: <value>`

Block level options override global options

Default values are Dita's defaults

The semantics of the options are

Option	Usage
<b>scale</b>	Scale factor specified as a decimal number (e.g. scale=1.5)
<b>antialias</b>	<b>true</b> to enable antialiasing; false to disable it
<b>separation</b>	<b>true</b> to leave spacing between adjacent blocks; false to remove it
<b>round-corners</b>	<b>true</b> to force round corners everywhere; false to use corner style from diagram
<b>shadows</b>	<b>true</b> to enable drop shadows; false to disable them
<b>debug</b>	<b>true</b> to display a debug grid in the diagram; false to remove it

# Chapter 3. Document Structure

## 3.1. Writing style

## 3.2. Document header

The document header is the first few lines of the documents. It can contain attributes that control the behavior of the converter at runtime or metadata about the document. The most common entries in the header are

- author - Author of the document
- email - The authors email address
- revdate - The date of the last revision
- revnumber - The current version number of the revision
- toc - Whether or not to use a table of contents

Various other attributes are available that contain information about the directories used, the behavior of headline numbering and so on. This is explained in the AsciiDoctor user manual.

The required attributes are described in the following parts of the guide.

## 3.3. Sections

### 3.3.1. Headings

Each document can only ever have one level 0 heading "=".

Typically when you want to reuse your contents elsewhere through includes you must start with level 1 "==" in any case. Wrongly constructed heading structure will produce warnings during build but more importantly can cause entire document structures to shift.

#### 3.3.1.1. Numbering

If you wish to use numbered sections please add the attribute ":sectnums:" in the header of the document.

## 3.4. Paragraphs

### 3.4.1. Newlines

Avoid adding newlines to structure the document. Your paragraphs should be contained by newlines

### 3.4.2. Line breaks

## 3.5. Inline Anchors

## 3.6. Cross References

## 3.7. Images in tables

In tables you should only use the inline image syntax with a single colon (`:`) (`image:myimage.png[]`). Block images will not work reliably.



(PDF export only) If your table row is smaller (less high) than the image you are using it possibly will not be rendered in an exported document. Either add additional content to the table row to make the text expand the height of the row or resize your image to fit the cell height. Single icons typically pose this problem.

## 3.8. Code in tables

## 3.9. Included files

You can include files. This can be achieved by using line number ranges or tags set in the external files. To include a file you will use the include macro (`+include::[]`). We will refer to this type of file here as "includes" this always means the document included not the one including.

Files that are supposed to be included should be designed to be handled as "includes" because of the way links are handled relatively if in an "include".

### 3.9.1. Heading Offset

You should design all your documents with the same structure as you would a top-level document. Start from the highest heading level and work down depending on the contents. When you eventually include one of these documents into an existing structure that already contains multiple headings you must adjust the level at which your "include" is injected into the existing order of document headings. For example your document looks like this:

*Ext-doc.adoc*

```
== This is my document

It contains some content that I wish to include.
```

And you have included it in this document:

*Master-doc.adoc*

```
== This is my master document

It already contains a lot of information

== But it should also refer to external info.

\include::Ext-doc.adoc[External info,leveloffset=+1] ①
```

① *leveloffset=+1* will move the heading level down 1 level to fit into the master document structure. Adjust this to your needs depending on where and how you included a document.

## 3.10. Ifdef and Ifndef Statements

### Conditional Preprocessor Directives

For some output types you will want to have special versions of the document generated. This can be achieved by building logical statements into the document that include or exclude parts of the file or attributes.

You can transform the structure and formatting of documents according to attribute values with this command. There are a lot of possibilities.

## 3.11. Restrictions and Tips

If a command or paragraph requires some special consideration or has a restriction, always place the note describing the restriction **before** the actual content. The user will process the instructions step-by-step and if a restriction is mentioned "after" the action there might be errors and misconfiguration.

### 3.11.1. Incorrect

Then copy that file to all servers as [/opt/autopilot/conf/topology.yaml](#) (directory must be created first.)

### 3.11.2. Correct



You must create a directory for the next command to work. Run `mkdir -p /opt/autopilot/conf/`.

Then copy that file to all servers as `/opt/autopilot/conf/topology.yaml`.

# Chapter 4. Formatting Instructions

## 4.1. Text formatting

For the "regular" formatting options (bold, italics, monospace etc.) please check out the [AsciiDoctor Syntax Quick Reference](#) since this covers these topics (and others) extensively.



This document will explain the style we have decided on to use in the Arago documentation and some less obvious usage patterns and options.



Many text elements allow for [Quoted Text Attributes](#) for even more powerful formatting options.

## 4.2. Special text formatting

Our instructions typically contain various special text elements that need to be communicated to the user.

### 4.2.1. Example

```
Please set the Port for ZooKeeper to "2181" in the file
'/opt/autopilot/conf/aae.yaml'. Make sure the attribute 'Allowed' is set.
```

Please set the Port for ZooKeeper to "2181" in the file [/opt/autopilot/conf/aae.yaml](#). Make sure the attribute [Allowed](#) is set.

## 4.3. Underline text

If you wish to underline a text you need to use this syntax:

```
This is my [underline]#underlined text#
```

This is my underlined text

## 4.4. Strike through

To strike through text you need to use this syntax:

This instruction is ~~[line-through]~~#valid for HIRO 5.4# no longer valid.

This instruction is ~~valid for HIRO 5.4~~ no longer valid.



Strikethrough is to be avoided as much as possible.

## 4.5. Special Characters

If you need to add emphasized characters to a description enclose them in curly brackets and the backtick inline syntax.

In some cases you must add a backslash (`\`) to escape characters in JSON.

In some cases you must add a backslash (\) to escape characters in JSON.



The syntax should be able to handle most special characters. If you need to add a special Ascii character please refer to the [Replacements syntax](#).

## 4.6. Warning boxes (Admonition)

The text boxes are called "Admonition" and you can find their syntax described here:

- [Quick Reference](#)
- [AsciiDoctor Manual](#)

### 4.6.1. Basic Usage

NOTE: This is a short note.

[TIP]

====

This is a much longer tip.

It can include line breaks and all kinds of [link:google.com\[content\]](#)

====



This is a short note.





This is a much longer tip.

It can include line breaks and all kinds of other **content**

### 4.6.2. Use of Admonition type

The different Admonition types are used to communicate with the user that a passage might be of special interest or contains additional information. Use admonition types only as described below to keep consistent across the content.

Admonition type	Usage reserved for
NOTE	General remarks. As the name suggests "notes" that the user <b>will want to know and needs to consider</b> .
TIP	Give tips how the user can <b>optimize their workflow</b> and for generally "not quite obvious" tips and tricks.
IMPORTANT	Use this block to point out <b>details that the user might overlook</b> that could lead to confusion. Mention important settings, files, procedures etc.
CAUTION	Use the Caution label to warn the user that certain steps, if not performed correctly, can lead to a <b>broken installation/configuration</b> . This should be reserved for cases that are (easily) <b>recoverable</b> .
WARNING	If a potential mistake can lead to <b>data loss or severe damage</b> to the installation (e.g. removing database tables, directories etc.) warn the user explicitly.

## 4.7. File names

Use the **backtick** (```) inline code syntax.

Please add the hostname configuration to ``/etc/hosts``.

Please add the hostname configuration to `/etc/hosts`.

## 4.8. Package names

The same applies for package names that are not yet present on the user system or must be downloaded/installed.

(.rpm/.deb/.zip etc.) use the backtick syntax.

```
Please install `your-package-1.4.2.rpm`
```

Please install [your-package-1.4.2.rpm](#)

## 4.9. Path names

Use the **backtick** (``) inline code syntax.

```
You can find the configuration files in `/opt/autopilot/conf/`.
```

You can find the configuration files in [/opt/autopilot/conf/](#).

## 4.10. Ports

Use the **backtick** (``) inline code syntax. The correct order for a port description is [PORT/PROTOCOL](#) e.g. [9443/TCP](#).

```
The reverse-proxy/load-balancer is working and can connect to `_9443/TCP_` on both nodes
```

## 4.11. Factory Names

Use the **backtick** (``) inline code syntax.

```
After retrieving all configuration settings, create the  
`DatatransformerInstance` and inject this into the constructor of the  
`CBProducerSender`.
```

After retrieving all configuration settings, create the [DatatransformerInstance](#) and inject this into the constructor of the [CBProducerSender](#).

## 4.12. Class Names

Use the **backtick** ( ``` ) inline code syntax.

To transform you messages into the process specific SDF you should implement your own ``de.arago.connectit.base.producer.DataTransformer``

To transform you messages into the process specific SDF you should implement your own `de.arago.connectit.base.producer.DataTransformer`

## 4.13. Function calls

Use the **backtick** ( ``` ) inline code syntax.

The method ``processSDF(SDF sdf)`` will be invoked every time a message has been received from the connector.

The method `processSDF(SDF sdf)` will be invoked every time a message has been received from the connector.

## 4.14. URLs

We need to distinguish between hyperlinks we want to point to the web or other resources and URL/URI/pURL that we need to show in their raw form.

### 4.14.1. External links

Please use the following format:

I want to show you `link:https://this-is-my-url.com/[my great website]`

I want to show you **my great website**



Add the "link:" macro and closing with the attribute list (even if it's empty) is important to properly render links inside the pages and flowing text.

### 4.14.2. Email links

The link is constructed like `mailto:address[Text, Subject, Body]`

If you have any questions please write me an `mailto:mnapp@arago.co[eMail, Contributor question, Hi I've got a question about hiro-documentation]`

If you have any questions please write me an `eMail, Contributor question, Hi I've got a question about hiro-documentation`

### 4.14.3. Raw URLs

There are many occasions where you will want to display a URL in its raw form. Use the `backtick` (```) inline code syntax. To avoid automatic URL parsing, add a `(+)` before and after the URL.

In your environment you will go to ``+https://my-hostname:8888/apps/+'``

In your environment you will go to `https://my-hostname:8888/apps/`



If you do not include the `(+)` sign it will generate a broken clickable link:

In your environment you will go to `https://my-hostname:8888/apps/`

## 4.15. Variables/Attributes

### 4.15.1. Variable/Attribute names

Use the `backtick` (```) inline code syntax.

### 4.15.2. Variable/Attribute values

Use italics to show values that need to be set.

Please set the ``Port`` to `__2181__`

Please set the `Port` to `2181`

### 4.15.3. Key:Value Pairs

If you want to display complete "Key:Value" pairs please use Italics and the backtick syntax.

```
Please set `__Port:2181__` in `/opt/autopilot/conf/arago.yaml`
```

Please set `Port:2181` in `/opt/autopilot/conf/arago.yaml`

### 4.15.4. Defaults

When describing variables or attributes of which a default value is known please include it in the description where it makes sense.

```
For a proxy setup the `Port` must be __7285__ (Default: __7284__)
```

For a proxy setup the `Port` must be 7285 (Default: 7284)

## 4.16. Images

There are two types of image links.

- Block image
- Inline image

For the correct display of images in exported documents it sometimes is necessary to use block image syntax.



For correct scaling of images in PDF documents you need to use block image syntax.

The normal inline image syntax looks like this:

```
image:/path/to/my_image.jpg/.png[width=?,my_image.jpg/.png]
```

Block image syntax simply add an additional colon (:) to the image macro.

```
image::/path/to/my_image.jpg/.png[width=?,my_image.jpg/.png]
```



AsciiBinder implicitly looks for images in a directory `images` in the same as the document. If you include a document in a different location the image must be copied to the images where it is included.



Always use the image filename as the alt text, this will aid in debugging missing images with our external tools.

## 4.17. Code

Code highlighting is done using the `pygments` library by default.



If you require a different highlighter for your document please check the [available options](#)

### 4.17.1. Code blocks

Please describe code blocks as best as you can. If you know the source language please add it to the `[source]` element.



Including actual AsciiDoctor source syntax for examples is somewhat tricky so please refer to the [Quick Reference](#) to see how to properly include source blocks.



You can add numbered comments to your code by using [Callouts](#)

### 4.17.2. Inline Code

Single lines of code should be enclosed in backticks. This can be used for lines standing alone or within flowing text.

```
To update your system run `yum clean all && yum update`
```

To update your system run `yum clean all && yum update`

## 4.18. Lists

### 4.18.1. unnumbered / unordered

#### Unordered lists

The correct format is:

```
* First entry
** First sub entry
* Second entry
** Second sub entry
*** Sub-sub entry
* Third entry
```

- First entry
  - First sub entry
- Second entry
  - Second sub entry
    - Sub-sub entry
- Third entry

### 4.18.2. *numbered / ordered*

#### Ordered lists

The correct format is:

```
. First entry
. Second entry
.. Second sub entry
. Third entry
```

A long description that is used to elaborate the third point but breaks the list.

```
[start=4]
. A fourth entry
```

1. First entry
2. Second entry
  - a. Second sub entry
3. Third entry

A long description that is used to elaborate the third point but breaks the list.

4. A fourth entry





The numbering and ordering syntax for lists is very powerful. Please refer to the [AsciiDoctor Manual](#) for advanced options.

## 4.19. Tables

### 4.19.1. Syntax

There are many syntax variants to create a table in AsciiDoc. This gives users a lot of flexibility but is also confusing. The preferred variant of the syntax is the "one line per cell" syntax.

The basic table syntax is described in the [Quick Reference](#)

For more elaborate formatting options (cells spanning multiple rows/columns, centered/aligned content etc) please refer to the [Advanced Syntax Options](#)

### 4.19.2. Titles

Please add a title to your table.

```
.Table of Ports
[cols="2"]
|===
|Port
|Description

|1234
|Used port for protocol x
|===
```

*Table 1. Table of Ports*

Port	Description
1234	Used port for protocol x

### 4.19.3. Headers

```
[options="header",cols="2"]
===
|First header
|Second header

|Content Row 1 Column 1
|Content Row 1 Column 2

|Content Row 2 Column 1
|Content Row 2 Column 2, content can be longer
and even contain multiple lines
|===
```

First header	Second header
Content Row 1 Column 1	Content Row 1 Column 2
Content Row 2 Column 1	Content Row 2 Column 2, content can be longer and even contain multiple lines

#### 4.19.4. Columns and formatting

The formatting options for rows and columns are very powerful and varied.

Please refer to the tables section of the [AsciiDoctor manual](#)



A great breakdown of common table formatting use cases can be found in the [Alternate AsciiDoctor manual](#)

# Chapter 5. Document Attributes

## 5.1. Header Attributes

### 5.1.1. Author

#### 5.1.1.1. Name

#### 5.1.1.2. Email

### 5.1.2. Revision Number

### 5.1.3. Revision Date

### 5.1.4. Free attributes

One example of the use of free examples is the [Correct linking for PDF](#) section.

### 5.1.5. Toggles

#### 5.1.5.1. TOC

#### 5.1.5.2. Section Numbering

## 5.2. Freetext attributes

### 5.2.1. Substitutions

You can substitute strings in your document by setting an attribute and referencing it in the text.

```
:myattribute: StringA  
I want to insert {myattribute} into this sentence.
```

I want to insert StringA into this sentence.

To make this work in tables or source blocks you need to add the [subs](#) option.

```
[subs="attributes"]
```

This is an example code block that uses {myattribute}

This is an example code block that uses StringA

# Chapter 6. Export to PDF

## 6.1. Install toolchain



This instruction is designed for installation on macOS. Installation under Linux based systems usually works through the provided distribution repositories. To install in Windows you will need to setup Ruby and it's dependencies yourself.

The main tool you will need is **AsciiDoctor-PDF** but you need to fulfill it's requirements.

Install **brew** by running:

```
/usr/bin/ruby -e "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Then install ruby by running:

```
brew install ruby
```

Once finished install the rest of the toolchain:

- AsciiDoctor
- AsciiDoctor-PDF
- Rouge



You will have to perform this operation as super user/admin.

```
gem install asciidoctor --pre asciidoctor-pdf rouge
```

## 6.2. Export a document



This will export the file with the default AsciiDoctor template.

To convert a document you then use:

```
asciidoctor-pdf yourdocument.adoc
```

### 6.2.1. Export with Arago template

If you need to generate a arago styled PDF document use this command and adapt your pathnames:



Replace `$HIRO_DOCS` with the path to the place you checked the Git project out.



Replace `your-file.adoc` with the file you wish to convert. The resulting PDF will be saved in the same location as the source file.

*makepdf.sh*

```
1 #!/bin/bash
2
3 asciidoctor-pdf \
4 --trace \
5 -r asciidoctor-diagram \
6 -a toc \
7 -a icons=font \
8 -a source-highlighter=rouge \
9 -a pdf-style=arago \
10 -a pdf-stylesdir=$HIRO_DOCS/_pdf-template/. \
11 -a pdf-fontsdir=$HIRO_DOCS/_fonts/ \
12 $1
```



If your document contains diagrams, add the "-r asciidoctor-diagram" parameter to process the diagrams.



The generated document will be created in the same directory as the input document.

## 6.3. Correct linking for PDF

The PDF export is based on the AsciiDoctor-PDF library which handles links differently than AsciiBinder. In order to be able to simultaneously have a document that can be used by both we must define some logic that rewrites certain links.

For the PDF all links within a document must be specified as a **relative path** from the document on which the PDF converter is called. For example:

```
install-config
├── standard
│   ├── setup-standard.adoc
│   ├── node-iam.adoc
│   └── node-database.adoc
└── verify
    ├── verify-database.adoc
    └── verify-engine.adoc
```

When you want to include ``install-config/verify/verify-database`` in `install-config/standard/setup-standard` for the normal version you would write:

```
include::install-config/verify/verify-database.adoc[Verify Database]
```

This will not work for the PDF converter since it can't find this file. Therefore you must define "When this is rendered by the PDF backend, change the link to". Which you can do like this:

```
ifdef::backend-pdf[]
:directory: ../verify/
endif::[]

include::{directory}verify-database.adoc[Verify Database]
```

Now *if* the PDF backend (backend-pdf) is detected the value of the free attribute "directory" will be changed to a relative prefix and create a working link for the PDF generator.

## 6.4. Images in PDF

Currently all images that are used in the PDF must be located in a `images` in the location where the `*.adoc` file resides that you wish to convert.



# Chapter 7. Maintaining the HIRO Documentation Repository

## 7.1. AsciiBinder

### 7.1.1. Distro names

When you add new branches to the `_distro_map.yml` file you must use the exact name of the branch as the "name".



Filenames and Branch names with three characters or less are not supported. A version branch `5.4` will not work. It will generate a string comparison error on build. Use at least four character names for everything.



Since version 0.1.10.1 distro names can no longer contain periods. The id `hiro-5.4` will give an error about invalid string. Only use letters.

```
hiro-online:
  name: HIRO
  author: HIRO Team
  site: main
  site_name: HIRO Documentation
  site_url: http://docs.hiro.arago.co/
  branches:
    develop:
      name: develop ①
      dir: develop
    5.4.0:
      name: 5.4.0
      dir: 5.4.0
```

① "develop" will be translated in the search index to: `<host>/<name>/<filepath>.html`. If this does not correspond to the actual branch file path the search index will be broken.

### 7.1.2. Topic Map



Filenames and Branch names with three characters or less are not supported. A version branch `5.4` will not work. It will generate a string comparison error on build. Use at least four character names for everything.

The topic map defines which pages are actually converted into HTML. Each file you wish to have as an

HTML file must be part of the `_topic_map.yml` file.

Filenames (before the extension) **must** be longer than three characters or they will not be recognized. A page `ki.adoc` will not work.

The topic map consists of a YAML structure that can have up to three levels of depth. It describes the projects document root broken down into directories (Dir) and pages. If you wish to list multiple files from within a directory you must use the "Topics" structure.

For example:

```
Name: Administration
Dir: administration
Topics:
  - Name: Java Installation
    File: java
  - Name: Config File Reference
    File: cfg-reference
  - Name: Network Time Server
    File: ntp
---
Name: Frontend
Dir: frontend
Topics:
  - Name: Cockpit
    File: cockpit
  - Name: KI Editor
    File: ki-editor
```

This will generate two main navigation points "Administration" and "Frontend" with their respective sub-points. Notice that three dashes `---` are the delimiter for a major navigation element. The filenames for the "File" entries must be exactly the file names (**without extension**) as they are stored in the project.

### 7.1.3. Generate Changelog

In order to generate the Changelog format run this command in the respective branch:

```
git log --pretty=format:"* %cd* - %s (%t)" --date=format:"%d-%b-%Y" --after
=<DATE> --simplify-merges | grep -v -e "fixup" -e "Merge pull request" -e
"Merge branch"
```

Adjust the `<DATE>` portion to the date of the last release so all following commits are tracked.

This will produce changelog messages of the format:

```
* *09-Jan-2017* - Switch path to UpdateSet (5c4df5e)
* *09-Jan-2017* - Update Automate manual (5f3add0)
* *06-Jan-2017* - Fix some links (1ff1c19)
```

### 7.1.3.1. Cleaning Changelog before publishing

Go through the results and fix malformed commit messages and remove internal information that should not be published to customers (e.g. other customer specific messages and changes).

### 7.1.3.2. Make available

Commit the changes into the file [general-information/changelog/<RELEASE>/documentation.adoc](#) (adjust [RELEASE](#) for the current release). Make sure the files [welcome/index.adoc](#) and your latest changelog meta file have the correct include to the latest file.

## 7.1.4. Special considerations for AsciiBinder

### 7.1.4.1. Include pages



The include behavior of AsciiDoctor is confusing.

If a page is included in the top level document it will need an absolute path in the macro.

If however an include statement is on a page that is itself included it will require a relative path. This can lead to some trouble. Keep this in mind when designing nested document structures.

**Explanation**

## 7.2. GitHub Best Practices

- Merge Strategy
- Reviews
- Labels
  - [github-labels.adoc](#)
- Squashing
- Cherry Picking

## 7.2.1. Commit Messages

Please write your commit messages in the following format:

- "Add new example to javascript element description"
- "Remove obsolete document for license connector"



Always include the place you have made changes e.g. "... in component x administration docs" so the customer can actively use the changelog to discover changes you made.

## 7.2.2. Files, filetypes and locations



Please refrain from adding very large binary files since GitHub is not designed to handle these efficiently. If your file is larger than 5MB please consider a different storage method or contact the project maintainer for advice.



All Pull Requests containing files larger than 5MB without sufficient reasoning will be rejected.

## 7.2.3. Pull Request Labels


### 7.2.3.1. Label Types







There are two types of labels. Branch or version labels and status labels.



Requests that have **on-hold** or **need-fixes** labels MUST NOT be merged!

*Table 2. Branch Labels*

Label	Description
	Indicates that the request should be merged into the current development branch.



Label	Description
 <b>5.4.x</b>	<p>Each release gets its own label. These are to be set to indicate into which branches the Pull Request should be merged to.</p> <div>  <p>If the next minor release is not yet defined create a .x branch and label.</p> </div>
 <b>5.4.1</b>	
 <b>5.4.2</b>	
 <b>5.4.3</b>	
 <b>5.5</b>	

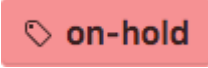



### Merging both develop and release branch

If larger changes are indicated to be merged into **develop** and a **release or version** branch, a thorough review is *required* to make sure that all changes actually apply to both versions.

The **develop** branch can contain major rewrites that will break consistency or give a wrong state of development if merged back into released versions without review.

Table 3. Status Labels

Label	Description
 <b>backport</b>	<p>This indicates that the requested changes must be backported to a previous version of the documentation. This can involve cherry picking of single commits to integrate changes that are embedded in larger rewrites. All requests with this label must be thoroughly reviewed and merged by hand.</p> <p><b>Only minor changes must be merged directly in GitHub.</b></p>
 <b>hotfix</b>	<p>This label indicates that a released version must be changed without a version bump. All affected versions must be listed with labels.</p> <p><b>If only specific versions are affected, indicate this in the comments.</b></p>

Label	Description
	<p>Some clarification is needed or other rewrites/circumstances block this pull request from being merged.</p> <p><b>Only when this label is removed can the merge label be set and actioned.</b></p>
	<p>Review has determined that some of the changes need to be reworked. This overrides the 'merge' label and means changes must not be merged.</p> <p><b>Optionally the 'on-hold' label can be set as well.</b></p>
	<p>Reviews have approved these changes and they can be merged into all indicated branches.</p> <div>  <div> <p>This label may only be assigned by the maintainer.</p> </div> </div>