Contributing to Magnolia Documentation

1.0, 2021-02-25: initial release

Table of Contents

Copyright

Revision History

Revision	Date	Comments
1.0	2021-02-25	initial release

Chapter 1. Introduction

Thank you for taking the time to contribute to Magnolia documentation. This page guides you through understanding the documentation workflow and contributing directly to the docs.

As of 2021-02-25, contributions are **internal** only.

A HTML version of this document is available.

Chapter 2. Clone the repository

The first thing you need to do in order to contribute to the docs is clone the repository.

Git needed

You must have git installed.

1. Navigate to https://git.magnolia-cms.com/projects/DOCUMENTATION/repos/product-docs/.
2. Clone the repo.
   a. Under **Actions** > **Clone**, choose **HTTP** since this is a public repository.

      If you want to use SSH, please read the Git SSH tutorial from Atlassian for assistance.

   b. Copy the URL.
   c. From your terminal, type `git clone` and paste the https://git.magnolia-cms.com/scm/documentation/product-docs.git URL as so:

      ```
      git clone https://git.magnolia-cms.com/scm/documentation/product-docs.git
      ```

   d. Hit Enter.

      You should see messages similar to the following:

      ```
      $ git clone https://git.magnolia-cms.com/scm/documentation/product-docs.git
      Cloning into 'product-docs'...
      remote: Enumerating objects: 1882, done.
      remote: Counting objects: 100% (1882/1882), done.
      remote: Compressing objects: 100% (1751/1751), done.
      remote: Total 22523 (delta 1108), reused 184 (delta 109)
      Receiving objects: 100% (22523/22523), 191.01 MiB | 13.62 MiB/s, done.
      Resolving deltas: 100% (14837/14837), done.
      Updating files: 100% (3756/3756), done.
      ```

Chapter 3. Repository structure

Go to the `/modules` folder to see the documentation. The documentation is structured as follows:

```
├──── modules/ ①
│     └──── HOME/ ②
│     └──── ROOT/ ③
│         └──── attachments/ ④
│         └──── examples/ ⑤
│         └──── images/ ⑥
│         └──── pages/ ⑦
│         └──── partials/ ⑧
│         └──── nav.adoc ⑨
├──── antora.yml ⑩
└────
```

Item	Description
1 `modules`	The required `modules` folder. This is where antora looks for the content during a build.
2 `HOME`	This is a stand-alone module that contains the landing page for the documentation site.
3 `ROOT`	The default **module** for antora documentation is `ROOT`. This can be changed. You can also have as many modules as needed to more easily managed docs.
4 `attachments`	Stores downloadable files such as pdf, csv, json, jar files, and so on. Anything that needs to be downloaded directly from the documentation website.
5 `examples`	This is typically used for code samples and where developers can directly either add or build triggers to pull information directly into the docs. *Example 1. Example* While updating `sample.java`, a developer also makes changes to the corresponding example. 1. `sample.java` exists in the `attachments` directory. 2. `sample.java` is used in 3 locations in the documentation with an include directive (`include::example$sample.java`). 3. All 3 locations are updated consistently from a single source.

Item	Description
6 `images`	Contains all images for the documentation. This includes videos and gif files.
7 `pages`	The deliverable pages for the site. Typically, pages contain multiple partials set as an assembly. For more information on assemblies, see Redhat's Modular Documentation.
8 `partials`	These are reusable bits of information that should be one of the following: • **concept** = labeled with the prefix `c` as in `c_services-cloud-intro.adoc`. Explains a concept or description of an item or process. • **reference** = labeled with the prefix `r` as in `r_variables.adoc`. Contains reference material such as definitions or variables. • **procedure** = labeled with the prefix `p` as in `p_installing-with-maven.adoc`. Guides users with a set of prerequisites and instructions. For more information on partials as modules, see Redhat's Modular Documentation.
9 `nav.adoc`	Establishes the navigation seen on the sidebar for this particular module.

10 `antora.yml`	Specifies important information for the documentation such as name of the repository, the version, and any potential attributes used throughout the documentation for this module.
	
	figure 1. antora.yml example
	
	`# release-related-page`
	`name: product-docs`
	`title: Magnolia CMS Documentation`
	`version: master`
	`nav:`
	`  - modules/ROOT/nav.adoc`

Chapter 4. Contribute

After you have successfully cloned the docs repository, you can contribute by adding new files or editing existing asciidoc files.

4.1. Prerequisites

- You need a text editor or IDE (such as Atom, IntelliJ, or VS Code).
- You must have git installed.

The first thing you want to do is identify what page you would like to edit. The best way to **find** the page is to use the URL from the docs site.

Site URL

https://docs.magnolia-cms.com/product-docs/Developing/Reusing-configuration/**YAML-inherit-and-include.html**

Page location in repo

modules/ROOT/pages/Developing/Reusing-configuration/**YAML-inherit-and-include.adoc**

4.2. Write

This section takes you through the steps to write and submit your work for your review.

Writing in asciidoc is simple. Plain text works most of the time.

You can refer to the Helpers file for tips on using and asciidoc and see the official Asciidoctor site for ore advanced uses.

1. Open your text editor.
2. Using your integrated terminal or your machine's terminal, navigate to the local directory where you cloned the doc repo.
3. Pull down the latest changes by running `git pull` while on the `master` branch.
4. Create a new branch by running `git checkout -b <JIRA>` using the JIRA ticket as the branch name.
5. Make your changes.
6. When ready, add your changes to the branch by running `git add .` - the **full stop** adds all local

changes.

7. Commit your changes by running `git commit -m "describe the changes"`.
8. Push your changes to your remote branch by running `git push origin <JIRA>`.
9. Go to https://git.magnolia-cms.com/projects/DOCUMENTATION/repos/product-docs/.
10. Create a Pull Request.

See Making a pull request for more details.

11. Add one of the doc team members for review. Currently, those are Julie Legendre, Ashraf Khamis, Martin Drápela, Alex Mansell.

4.3. Review

The review process follows a typical software development cycle flow. After the Pull Request is created and you have assigned reviewers, reviewers can comment, assign tasks, approve, and reject the request.

The Magnolia documentation team maintains the site and has the final say on merging the content into the official product documentation.

4.4. Contribution templates

Currently, there are templates for writing modules and release notes.

See the product-docs-templates repo for these templates.

Chapter 5. Site build

The content in this repository is fetched via the playbook (*1 below*) in the antora build we have set up for Magnolia documentation.

```
site:
  title: Magnolia CMS Docs
  # the 404 page and sitemap files only get generated when the url property is set
  url: https://docs.magnolia-cms.com/product-docs/

...

content:
  edit_url: ~
  sources:
...
  - url: https://git.magnolia-cms.com/scm/documentation/product-docs.git ①
    branches: [master]
    start_path: docs
...
```

5.1. View your changes locally

While building the site locally is a great way to see changes, the documentation team are able to view your pull request locally when we review.

In light of this, you should only build the site locally if you have the following installed and are accustomed to using them:

- git
- npm
- gulp
- antora

1. Clone the antora site repo.
2. Run `npm install`.
3. Cd into the `ui` folder.
4. Run `npm install`.
5. Run `gulp bundle`.
6. Cd back to the root folder.
7. Copy the content from `playbook.yaml`.
8. Create a new file called `local-playbook.yaml`.
9. Paste the content from `playbook.yaml` into `local-playbook.yaml`.

10. Under **content** > **sources** in your `local-playbook.yaml` file, point the playbook to the local path that holds your product-docs repo.
11. Set the URL as the path relative to your local antora site repository.
12. For `branches:`, set it to `HEAD`. This ignores the worktree and shows you content from whatever branch you're on.

figure 2. Local Playbook

```
content:
  sources:
  - url: ../product-docs ⑪
    branches: HEAD ⑫
    edit_url: false
```

13. Under **output** in your `local-playbook.yaml` file:

```
output:
  dir: ./local-build
```

14. Run `antora local-playbook.yaml` and open `./local-build/index.html` from the `local-build` folder.

It's best to make an alias or script to run this. For me (Alex - on Windows), I have created an alias called `local` so I just need to type it into my terminal and the work is done for me. See Creating bash aliases for more details.

For example: I have an alias named `local` that navigates to my `antora-site` folder, runs the local playbook, and then opens the output in my browser as shown below:

```
alias local="cd /c/Users/aubur/Documents/Magnolia/docs-as-code/antora-
site/; antora local-playbook.yaml; start chrome local-build/index.html"
```

Example 2. Why not just use the build/site folder?

As a precaution, it's usually best practice to avoid rebuilding into this directory with local changes.

5.2. Build environments

There is both a staging and production environment for Magnolia Documentation. We use S3, AWS, and the documentation toolchain to produce this. Every pull request triggers a staging build. This can be viewed at on the staging site.

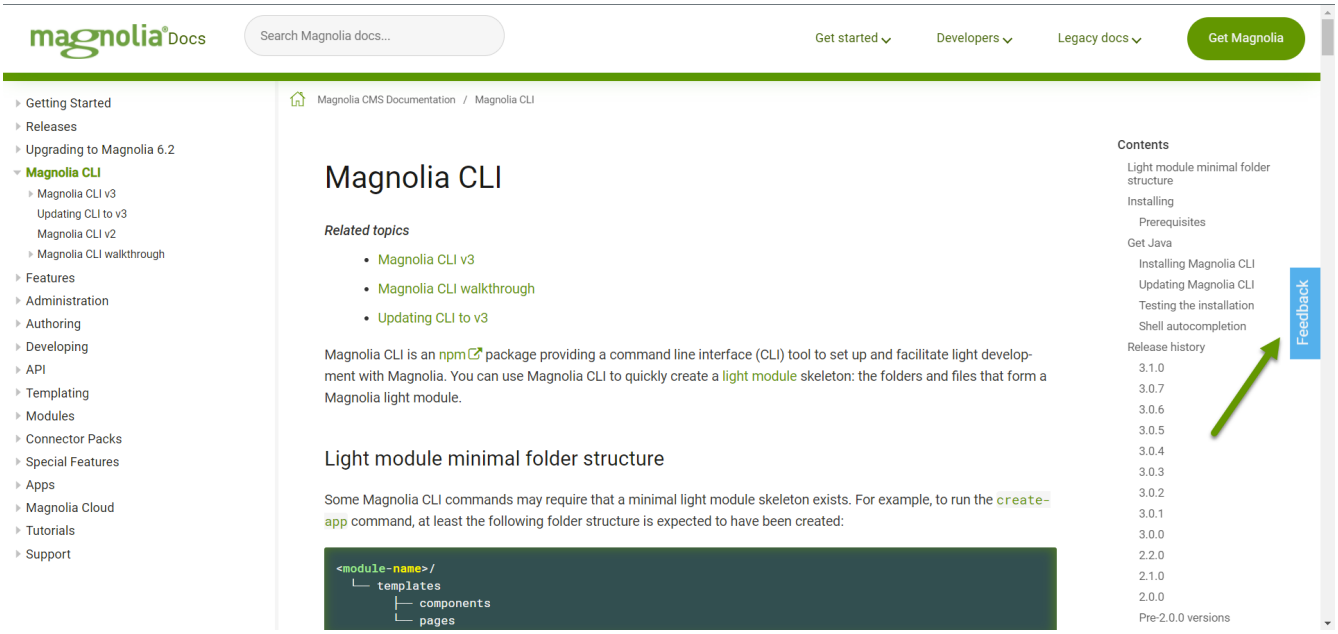
View the official Magnolia Documentation site.

I want to learn more

See the asciidoctor and antora official sites for more information on these tools.

Reporting an error or bug

You can report errors, doc bugs, and enhancements for Magnolia documentation using the **Feedback** button found on the right side of the documentation site as shown below. This takes you to the **DOCU** Jira project for Magnolia where you can submit your request.



DOCU ticket tips

Use a clear and descriptive title

For example, “Installing Magnolia provides the wrong minimum Java version”

Describe the problem or enhancement in full

Describe the issue in detail, including any expected results and how they differ from the actual results. The documentation team are not necessarily experts in all aspects of Magnolia. Providing a detailed description along with expected results will greatly reduce the turnaround time.


magnolia®