Documentation User Guide

How to contribute to the documentation repository

Table of contents

Creating new pages

The first step on adding new content is to understand the Documentation structure. Then, proceed with Adding content.

Content templates and guidelines

Templates and guidelines for specific types of content are available:

Evaluation Boards

Evaluation boards pages compile all content related to an evaluation board, from the hardware overview, features, kit contents, hardware, user guides, developer guides, and source code.

A top-level template is available at docs/contributing/template/example/eval.rst (rendered). Please pay attention to the comments only visible on the source code, and remove them as you follow the template.

Base page structure

Evaluation boards pages are divided into five main sections:

- Overview
- User Guide
- Developer Overview
- Developer Guide
- Help and Support

The evaluation board begins with an overview of the board, similar to the first paragraph at analog.com.

The Overview section provides key points about the evaluation board to help the user understand it and choose the ideal solution (baremetal or Linux?, hard core or FPGA-based?, etc.).

Then, cover key features, evaluation board kit contents, required equipment, hardware overview (all generic to all carriers).

The “User Guide” section contains guides aimed at users, including those without deep technical knowledge. Start with the most plug-and-play solution, gradually diving deeper into technical aspects. If something is too technical, save it for the “Developer Overview” or “Developer Guide” sections.

The “Developer Overview” section should provide all technical details that a developer may want to know and summarize and link all source code and documentation available. It’s essential to leverage the Git role and In organization reference. Avoid using full URL paths, as they make consistency checks difficult.

Similar to the “User Guide” section, the “Developer Guide” section contains guides but aimed at developers! Here is where you can really dive deep into the details and complex parts. However, make sure not to duplicate content already present on other pages, especially external ones. For example, the no-OS drivers’ source repository already contains a page explaining the driver details. These developer guides should focus on intricate details of using the evaluation board with the solutions.

Finally, the last section “Help and Support” contains generic information and links, but you can still add help/ support information related to your particular evaluation board.

Simplify the base structure

Some evaluation boards are simpler than others, and may not require splitting content into subpages.

If the evaluation board in question does not contain or need a developer guide, then the “Developer Guides” section can be removed, and the “Developer Overview” renamed simply to “Developers” with pointers to the source code.

Additionally, if there is only one carrier and user guide, the user guide can be integrated into the main page, replacing the carrier-agnostic paragraphs with the carrier-specific details.

An example of this structure is eval-adxl355-pmdz. A more complex example is eval-ad4052-ardz.

Structure Rationale

This structure aims at minimizing content repetition, clearly defining the boundaries between “Plug&Play” solutions and highly-technical developer resources, and ensuring consistent linking to the source code and related documentation.

It also enables generating custom documentations with only the pertinent content for the evaluation board, such as using Doctools’ Custom Doc with the following YAML file:

```
project: "EVAL-AD4052-ARDZ"
description: "Evaluating the AD4050/AD4052 Compact, Low Power, 12-Bit/16-Bit, 2 MSPS Easy Drive SAR ADCs"

include:
  - documentation/eval/user-guide/adc/ad4052-ardz
  - documentation/linux/drivers/iio-adc/ad4052
  - hdl/projects/ad4052_ardz
  - no-OS/drivers/ad405x.rst
  - no-OS/projects/ad405x.rst

entry-point:
  - caption:
    files:
      - documentation/eval/user-guide/adc/ad4052-ardz/index.rst
  - caption: HDL Design
    files:
      - hdl/projects/ad4052_ardz/index.rst
  - caption: Linux IIO Driver
    files:
      - documentation/linux/drivers/iio-adc/ad4052/index.rst
  - caption: no-OS Driver&Project
    files:
      - no-OS/projects/ad405x.rst
      - no-OS/drivers/ad405x.rst
```

Produces a concise and resourceful user guide.

Documentation structure

This repository hosts any type of content that is not version controlled with a particular source code, or in other words, “don’t deserve their own repository”.

Due to this, there are multiple topics, for example, there is information from Linux drivers to Evaluation boards user guides.

As an analogy, if this documentation were a encyclopedia, each topic would be a volume.

To create each “volume”, two toctrees replicate the structure of the context top level.

For example, while in docs/index.rst#L24 we have:

```
.. toctree::
   :caption: Linux Kernel & Software
   :maxdepth: 2
   :glob:

   linux/*/index
```

At the specific context toctree (docs/linux/index.rst#L9) we have:

```
.. toctree::
   :glob:

   */index
```

Glob is used to match any document that matches the pattern and avoids simple but annoying merge conflicts of contributors adding pages to the same toctree at the same time.

Use case for the structure

This structure enables to concatenate other documentations (“volumes”) to this one, allowing to generate an aggregated monolithic output, for example.

Suppose we have a repository called `my-repo` with the following toctree:

```
.. toctree::
   :glob:

   */index
```

Tip

Notice the usage of the `:glob:` options. It is particular useful to avoid merge conflict scenarios.

To add to this doc, we only need to append to *docs/index.rst* as:

```
.. toctree::
   :glob:

   my-repo/*/index
```

And copy `my-repo/docs` as `documentation/docs/my-repo` (mostly).

Adding content

The documentation is highly hierarchical and contextual, that means a page about “Peeling Blue Bananas” should be located at `fruits/banana/blue/peeling.rst` and **not** `eval/tutorial-peeling_blue_bananas.rst`.

The title should also be kept short, since it directly inherits the context from the hierarchical structure, so it’s preferred:

```
Peeling blue bananas
====================
```

Over:

```
Fruits tutorials: peeling blue bananas
======================================
```

At the `toctree`, the title shall be overwritten to reduce the title length on the sidebar further:

```
Blue bananas
============

.. toctree::

   Peeling <peeling>
   Recipes <recipes>
```

Tip

Don’t overthink the location of the content, it can be easily moved later. Just try to keep it *contextual* and *hierarchical*.

Having that in mind, proceed with creating the directories, toc-entries, and files for your content:

```
$ cd ~/documentation/docs ; pwd
~/documentation/docs
$ mkdir my_topic
$ # Add "My Topic" to the main index
$ vi index.rst
$ # Create topic/volume index
$ vi my_topic/index.rst
$ # Create more content
$ vi my_topic/page0.rst my_topic/page1.rst
$ # Add/create images
$ cp ~/some-image.svg my_topic/
...
```

Edit *my_topic/index.rst*, adding a title and some content.

Build the doc and see the changes:

```
~/documentation/docs$ make html
```

Tip

Sphinx only rebuilds modified files, so toctree changes may look like they are not “applying” to the documentation. Just rebuild the whole doc with `make clean html` if the output is confusing.

Even better than having to run `make html` at every edit, you can leverage Why was Author Mode renamed to Serve? to have a live-updating instance of the doc, you just need to save the file and the build will be triggered automatically.

Importing from DokuWiki

To import content from dokuwiki, a script is available to help on this task: DokuWiki to Sphinx (bash.sh).

It requires you have `pandoc` and `sed` installed:

```
sudo apt install pandoc sed
```

It will try its best to reduce the amount of manual work necessary, still, please review the content carefully.

For images, ensure to click on the image on *wiki.analog.com* to ensure you download the original and not the compressed image.

Always prioritize vector images (*.svg*).

Finally, content yet not imported, keep/use the Dokuwiki role. And for deprecated content, add the qualifier `+deprecated`, for example:

```
:dokuwiki+deprecated:`Old content <resources/old/content>`
```

The reason for this is that with this differentiation we can easily track yet to import pages and deprecated content with:

```
~/documentation/docs$ grep --exclude-dir=_build -rnw :dokuwiki:
software/libiio/internals.rst:58:like :dokuwiki:`GNU Radio ...
software/libiio/index.rst:270::dokuwiki:`here <resources/t ...
...
~/documentation/docs$ grep --exclude-dir=_build -rnw :dokuwiki+deprecated:
```

```
software/libiio/index.rst:54:* :dokuwiki+deprecated:`Beac ...
...
```

Documentation guidelines

This documentation is built with Sphinx and all source code is available at the path docs.

To contribute to it, first read Forking and publishing, read the guidelines (both the general **and** the additional guidelines below) and also Creating new pages.

When you are satisfied with your contribution, open a pull request with the changes to this repository.

Templates

Any page can be used as a template.

In particular, for evaluation board user-guide, use the ADRV9009 & ADRV9008 pages as a template.

For (future) template pages with `:orphan:` on the first line, remove it. This marker is used to hide the templates from the TOC tree.

Also, instructions using the comment syntax may be present on the page and also need to be removed. Those comments have the format:

```
..
   I'm a comment
```

Forking and publishing

The steps below are a walk-through to contribute to this repository, It ensures that GitHub Actions and GitHub Pages are enabled, so you can run continuous integration and see the pages live at *<your_user>.github.io/documentation*, and git-lfs artifacts are properly synced.

> Note
>
> Using the Python virtual environment (venv) is recommended, but if you wish to not use it, skip steps in green.

Preparing your origin

There is three options to host your work, for users:

- Fork: that want to use the GitHub flow (recommended).
- Copy: that want to work privately first.
- Branch: with write access to *analogdevicesinc* organization.

Fork

Ensure git-lfs is installed with:

```
sudo apt install git-lfs -y
```

Fork the *analogdevicesinc/documentation* repo on your account.

Enable the workflows on the forked repo at *github.com/<your_user>/documentation/actions* by clicking the green button

"I understand my workflows, go ahead and enable them".

Clone the repository:

```
~$ git clone https://github.com/<your_user>/documentation \
     --depth=10 -- documentation
~$ cd documentation
```

Copy

Ensure git-lfs is installed with:

```
sudo apt install git-lfs -y
```

Clone mainland:

```
~$ git clone https://github.com/analogdevicesinc/documentation \
     --depth=10 -- documentation
~$ cd documentation
```

Setup both origins, for example, call *analogdevicesinc* `public` and your copy `private` at the *.git/config*, similar to:

```
[core]
      repositoryformatversion = 0
      filemode = true
      bare = false
      logallrefupdates = true
[remote "public"]
      url = https://github.com/analogdevicesinc/documentation.git
```

```
     fetch = +refs/heads/*:refs/remotes/public/*
[remote "private"]
     url = https://github.com/<your_user>/documentation.git
     fetch = +refs/heads/*:refs/remotes/private/*
[branch "main"]
     # Set your private copy as upstream
     remote = private
     merge = refs/heads/main
```

Push the working branch to your copy.

```
~/documentation$ git push private main:main
```

Fetch from *analogdevicesinc* and push to your copy the large files binaries:

```
~/documentation$ git lfs fetch --all public
~/documentation$ git lfs push --all private
```

Branch

If you have write permission to the repository, you shall add your work to a branch at mainland, then just:

Ensure git-lfs is installed with:

```
sudo apt install git-lfs -y
```

Clone the repository

```
~$ git clone https://github.com/analogdevicesinc/documentation \
    --depth=10 \
    -- documentation
~$ cd documentation
```

Create and checkout a branch

```
~/documentation$ git checkout -b <your_branch>
```

Preparing your environment

Clone and build the doc for the first time (working directory: repo root):

Ensure pip is up-to-date:

```
pip install pip --upgrade
```

Setup the virtual env at the repo root path:

```
~/documentation$ python -m venv ./venv
```

Activate the virtual env:

```
~/documentation$ source ./venv/scripts/activate
```

Install the requirements:

```
~/documentation$ (cd docs ; pip install -r requirements.txt --upgrade)
```

Build the doc (output at docs/_build/html):

```
~/documentation$ (cd docs ; make html)
```

Adding content

Add a new topic and pages (working directory: docs).

On *index.rst*, add a new topic:

```
.. toctree::
   :caption: My new topic
   :maxdepth: 2

    my_topic/index
```

Or add to an existing, for example, in *eval/index.rst*.

> Tip
>
> Don't overthink the location at this point, it can be easily moved later.

Create a new folder and file matching the entry from last step:

```
~/documentation/docs$ mkdir my_topic; touch my_topic/index.rst
```

Edit *my_topic/index.rst*, adding a title and some content.

Build the doc and see the changes.

Commit the changes.

For a extensive guide on adding content see Creating new pages.

Pushing and triggering the CI

The CI (.github/workflows/top-level.yml) builds the doc and pushes to the `gihub-pages` branch and is triggered on push to main and on pull request (every time):

- On pull request, the build doc target is run, which builds the doc and stores it as an artifact.
- On push to main, the build doc and deploy targets are run, the latter commits the doc artifact to the gh-pages branch.

> Tip
>
> You can see the runs at github.com/<your_user>/documentation/actions.

Enable GitHub Pages to have the public website configure GitHub Pages at *github.com/<your_user>/documentation/settings/pages*:

- Set Source as "deploy from branch"
- Set the branch as "gh-pages"

Resuming work at a later time

Reactivate the virtual environment with:

```
~/documentation$ source ./venv/scripts/activate
```

Ensure the tools are up to data from time to time with:

```
~/documentation$ (cd docs ; pip install -r requirements.txt --upgrade)
```

Edit, build, commit, push as usual.

Understanding git lfs

Since git lfs is not that common in the wild, it may be tricky to get the hang of it.

First of all, the basics: lfs replaces binaries files with pointers, and stores the binaries outside the git repository, in an external server.

When you do `git clone/pull`, by default lfs will also download the binaries at the “smudge” step. You can change this behaviour by setting globally `git lfs install --skip-smudge` or temporally with `GIT_LFS_SKIP_SMUDGE=1` environment variable.

If during a clone or pull you obtain the error:

```
Encountered n file(s) that should have been pointers
```

That simply means that someone pushed files to remote that should have been pointers (defined in the *.gitattributes* file). And to fix is simple:

```
~$ git add --renomalize .
~$ git commit -m "Convert binary files to pointers"
~$ git push
```

And advise the committer to ensure he has git lfs enabled with `git lfs install`.