HPC-BP Webinar Q&A

# One Good Tutorial: Defining a "Minimum Viable Documentation Product" for Scientific Software

Date: 18 March 2026
Presented by: Peter K. G. Williams (Center for Astrophysics, Harvard & Smithsonian)

(The slides are available via the link in the page's sidebar.)

---

***Q:** Is it best to have a separate person responsible for documentation, or should the team that develops the software be responsible for writing the documentation?*

**A:** This is a good question — there are factors pushing in both directions.

Generally, the team that develops the software will understand it the best, and in my view that's a supremely important thing when it comes to writing documentation. They will also be positioned to embrace the docs as code approach that I recommend for most projects. But there's of course the risk of knowing the software *too* well, and having difficulty putting yourself in the shoes of someone who's just coming to it for the first time. It may also be true that there are scientist subject-matter experts (SMEs) associated with the project who have important, relevant knowledge but aren't developers.

In most cases, I think it will turn out to be best for the development team to be responsible for writing the bulk of the documentation. But if at all possible, time should be set aside for user testing, or at least review, by one or more people external to the team. This is especially important for the tutorial, which is most squarely aimed at newcomers to the project.

***Q:** How do you make good documentation, keeping multiple users in mind? Documentation about setting up access, could have different steps and routes depending on where the users are coming from. And we wouldn't want to bombard users with all the information.*

**A:** This can be a real challenge. One approach I like is to create different landing pages or entry points into the documentation aimed at different user profiles — developers, administrators, etc. You can think of it as if the separate landing pages are presenting different "views" of the same underlying documentation corpus, with different emphasis or sequencing depending on the user profile.

Another approach, which can dovetail with the above, is basically to write the same documentation multiple times with different user types in mind. As a developer I hate this kind of redundancy, but if you're confident that you're going to keep everything in sync, it can be the most pragmatic approach. One can imagine that with some engineering work, you could use LLMs to automate this kind of tweaking.

***Q:*** *Do you have suggestions for when a team has multiple documentation sources that might overlap? (For example, some internal documentation, some user facing documentation some administration documentation but for different team, etc.)*

**A:** This is another challenging situation. One early question to ask is: can we consider migrating all of these sources into a new, unified corpus, or do we have to keep them separate? In most cases, you probably have to keep them separate. But if not, perhaps you can solve the problem by merging everything! Once again, LLMs may be of help here.

Presuming that you have to keep your documentation sources separate, the next thing to consider is whether you want to try to make them *look* unified to the outside world. All else being equal, this is probably the preferable way to do things (it's nice to have all of your docs in one "place"), but depending on the document formats and a bunch of other factors it may not be worth the technical lift. If the circumstances are favorable, it might not be too hard to set up an automatable process to gather the input materials and merge them into a relatively coherent-looking output. (For instance, if you have HTML API docs generated via several different language-specific tools, you can probably merge them into one large static HTML website.) Once you have a first version of a merged product, the most value comes from (1) providing useful navigation of the *total* documentation corpus and (2) enabling cross-linking *between* the different source document types.

Sometimes it's simply not going to be practical to create a unified facade for your various different sources. In those cases, however, my priorities would be the same: provide useful navigation of the total corpus, and try to make it easy to cross-link between different document categories. In this situation, the different sets of documents might manifest as separate websites on separate domains, and so the question of cross-linking depends critically on setting up easy-to-use, link-friendly URL structures within each set of documents.

***Q:** When a document has multiple hierarchies, how do you best represent this? Would you recommend a flat hierarchy (long single pages) or multiple level, interlinking docs (short pages per topic)*

**A:** In general, I strongly recommend flat hierarchies. In my experience, the Python maxim “flat is better than nested” applies much more widely than just that one language. I’ll note that this may tend to encourage the creation of longer single pages, but in principle that’s a separate matter — you can have long articles that are organized in a deeply nested way, or many short articles that are organized in one big alphabetical list.

But, the more I work with documentation, the more I want to get away from hierarchical organizations altogether. Consider Wikipedia: its millions of articles aren’t organized into any overarching hierarchy; they just live in a big amorphous “bucket”. And yet it’s clear how to find what you’re looking for: you search for it, and you follow links. With sufficiently good search and cross-linking, I think that the need for hierarchy recedes into the background quickly.

That being said, most digital documentation tools and static website generators bake in, at a fundamental level, the idea that your documents must be organized into some kind of hierarchy. I suspect that we will need an entirely new class of tools (much more like wikis) to get away from this paradigm. Until we have those, I recommend keeping things as flat as you can get away with. The beauty of hypertext is that within your primary hierarchy, there’s nothing stopping you from creating “index” pages that present alternative organizational schemes as needed.

***Q:** How might the use of AIs for RAG (retrieval augmented generation) and documentation and example generation lead to adjustments into the guidelines presented?*

**A:** Indeed, you’ll note that I didn’t mention LLMs in my presentation at all. But I think that these will transform our notion of “documentation” rapidly. In particular, I wouldn’t be surprised if sooner rather than later, the landing pages of documentation sites evolve to basically be LLM chat prompts instead of hierarchical tables-of-contents. It seems plausible that RAG techniques can be made to work reliably enough such that we’ll be able to throw all of our documentation into a bucket on the backend, and the frontend will synthesize everything on the fly in dialog with a user.

Separately, you can imagine using LLMs to help create documentation source materials, such as example generation as mentioned in the question. I haven't tried this, but LLMs would probably also be excellent at copyediting one's prose to steer it towards a highly uniform, best-practices technical writing style. I think that the concerns here are for the most part the same as you would encounter in any other LLM-assisted writing activity. In my view, it's of paramount importance for there to be a "human in the loop" reviewing and taking responsibility for the final product. And when it comes to scientific software, I'd like to think that the humans behind it are the ones that will best understand what's novel and interesting about it, and think that means they should be the most engaged in writing the most important pieces of documentation (tutorial, synopsis, etc.).

***Q:*** *Any thoughts on writing documentation* for *LLMs to consume?*

**A:** I don't have experience in this area, so I can't offer anything very specific. But I can say that I think that this is going to be an increasingly significant aspect of what we do when we "do" documentation. Earlier, I mentioned the idea that the landing page for your documentation website will likely evolve into an LLM chat prompt. But in that scenario, people are likely going to be cutting out the middleman and asking their text editor or agent AIs about *your* project. Like it or not, part of your job will be to help ensure that those people get accurate, helpful answers.

I believe that there are emerging conventions about how to supply this information, such as `llms.txt`. But of course this field is incredibly fast-moving, so we'll probably all have to stay on our toes for a while and aim at a moving target.

***Q:*** *What are tips and techniques to make the documentation in sync with changing workflow? Especially when the documentation involves multiple moving components (different hardware and software )*

**A:** The single best thing I can recommend is adopting a docs as code workflow. It by no means solves every problem, but in my experience, it makes a huge difference to have the relevant documentation located as "close" as possible to the thing being documented.

Doctests can also be an important piece of the puzzle here. That is, automated testing that processes your documentation and checks its validity. The most common form of doctesting focuses on code samples, and making sure that they actually work, or at least that they compile correctly. But you could imagine setting up infrastructure to automate checking other elements like your installation instructions. This is one reason to author narrative docs such as tutorials as Jupyter notebooks: it's pretty easy to automate their execution.

I haven't seen this done, but you could also imagine setting up CI rules and lints to try to help keep things in sync. For instance, if a certain source code file is modified in a pull request, you have a check that demands that an associated documentation source file must also be modified.

***Q:*** *Are there ideas for making documentation easy for non coders to contribute?*

**A:** I think this is a surprisingly thorny question. I've mentioned the docs as code approach several times and I think that on balance it is the best approach for the vast majority of projects. But it does mean that to contribute to the documentation, one needs to be comfortable working with developer tools like Git, and this can be a real barrier. In the specific case of scientific software, there might be subject-matter experts (SMEs) who could contribute significantly to the documentation but aren't going to be comfortable using version control tools or code editors.

You can imagine "hacks" to enable contributions from such people: maybe they can write narrative docs in a standard word processor, and a developer can adapt their contributions into a pull request written in Markdown.

This kind of approach has obvious downsides, but as far as I'm aware, better methods generally demand a lot more engineering effort. The command-line accounting software Beancount has a system where the docs are authored in Google Docs but automatically synchronized to a static Markdown site — neat, but probably a lot of effort to administer. In extreme cases you might partition your docs into one corpus maintained "as code" and another maintained with WYSIWYG tools friendlier to non-developers.

I believe this is another area where options are somewhat limited by our documentation authoring tools. There's neat work being done with technologies like CRDTs that allow for real-time collaborative editing like Google Docs, but with a well-specified data model that you can save to disk and integrate with developer workflows — see Patchwork, for instance. I think this could eventually be the basis for tools that provide the benefits of "docs as code" along with the ease-of-use of WYSIWYG, collaborative tools.