HOW TO USE STATIC WEBSITE GENERATORS TO BUILD TECHNICAL DOCUMENTATION

Table of Contents

I. Introduction

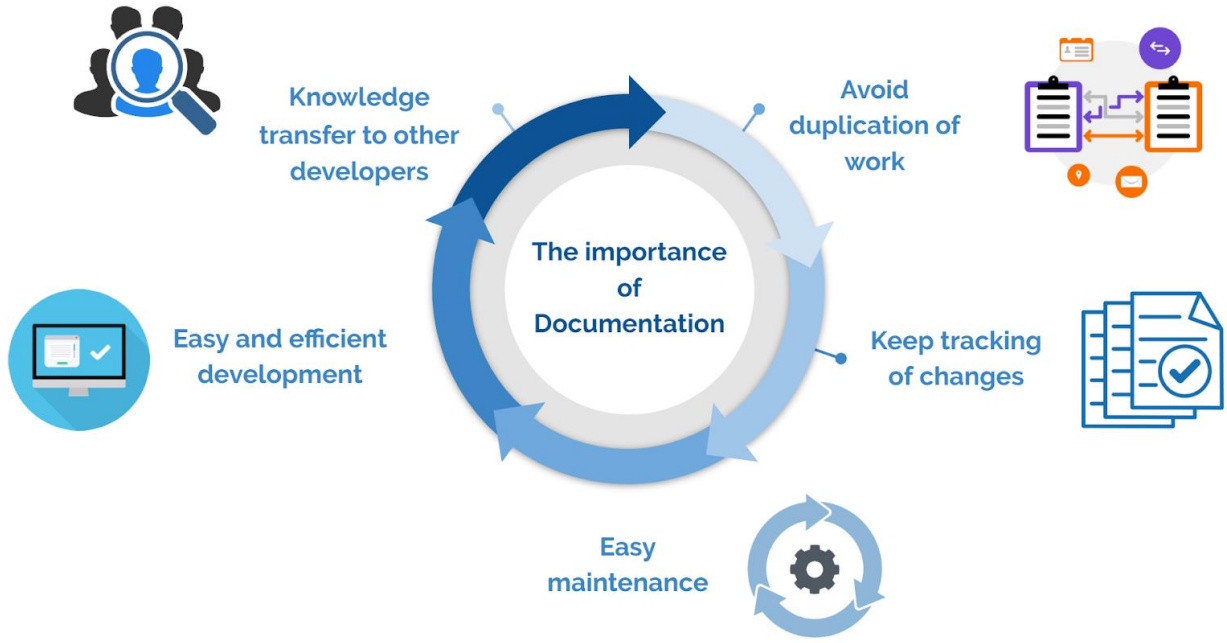
Technical documentation is an invaluable resource for projects and software systems. Yet, identifying the ideal documentation tool, setting up the appropriate process while keeping technical project documentation up to date, making it accessible and being efficient about it can be a challenge.

I.1. What is technical documentation?

In simple terms, technical documentation is any document that explains the use, functionality, creation, or architecture of a system. The key to writing good technical documentation is in the format of the document. Technical documentation isn't just about capturing information. It's about presenting it in a way that's easy for the reader to understand and use.

I.2. Why is technical documentation so important?

The presence of documentation helps keeping track of all aspects of a system and improves its quality over time. Its main focus is easy and efficient development, as well as maintenance and knowledge transfer to other developers. Successful documentation will make information easily accessible, helps to avoid duplication of work, simplify the product and to cut support costs.



A screenshot illustrating the importance of system documentation

I.3. Our case study: documenting code for one of our partners' project

CartONG's partner had set up a project for monitoring economic inclusion programs. The monitoring system had been developed using Google Apps Script which helped with data flow integration from KoboToolbox, by pulling the data collected from its API to Google spreadsheets for analysis and data validation before, finally, publishing to the program's open data platform.

Part of the work conducted by CartONG's teams since late 2018, was to find a tool for this partner that would allow collaborative editing between the project's developers, putting all the system's code in version control and making the documentation process truly agile.

II. Identifying the right tool to produce good technical documentation

There were several steps involved before we could identify a tool that would help the technical documentation of this project to be clear, consistent and easy to read.

II.1. Step 1: Identify reasons why a new documentation tool is needed

After an initial assessment, we realized that the existing documentation was not fully complete, missing important code components and centralized in a Google document; therefore, making it difficult to maintain developers' documentation as there was no system in place that would allow keeping track of every code modification while applying these direct changes to the system documentation. The existing documentation also made it difficult to involve more developers due to the large amount of time needed to learn and understand the code. Last but not least, it was not easy for a new developer to easily replicate the system for tests or for a system evaluation.

II.2. Step 2: Research existing documentation tools and select the most appropriate one

In order to write a good technical documentation, you need to use the right software documentation tools. Basically, a documentation generator is a programming tool that generates software documentation and supports people in generating HTML within the source code. One of the hardest part is of course to identify which tool will be suitable for one's project.

II.2.A. Identify a markup language suitable for your project

There are several documentation generator tools that can be used to write a well-structured technical documentation. These tools have different writing methods and markup languages which include Markdown and reStructured Text. The type of markup languages serves as the basis for many documentation systems and plays an important role in choosing a particular tool.

Markdown is the most popular and widely used tool, even though restructured Text has more features, is better standardized and more uniform and, as such, is perfectly suited for complex projects. Markdown is mostly used for adding comments, for to-do lists, for creating simple web-pages and writing blog posts, but it doesn't have a standard extension mechanism while restructure Text has built-in support for extensions.

Which one should you choose? This depends on your project. For large and small software/web-development projects, it makes more sense to choose reStructured Text. Using this method for writing documentation saves time as you can use themes that allow you to change the styling of the documentation thus controlling how the documentation is formatted. You also do not have to think about rendering of the documentation while writing it, you can just focus on the content and code snippets.

II.2.B. Identify the hosting platform

One of the main aspects to consider is where to host your documentation. Identifying a hosting platform that is easy to use and maintain, compatible with your system's code programming language and accessible by your colleagues are some of the criteria that were used to identify the best tool for our partner's project. The most popular hosting options include: GitHub (& GitHub Pages), Read The Docs and Dropbox Paper.

The following table compares technical information for a number of documentation generators:

- Operating system support (Windows, Mac OS X)
- Language support (some of programming languages that generators recognize)
- Output formats that generators can write
- Other features (possibility of extended customization, generated diagrams, etc.)

Tool	Operating system		Language Support	Output formats		Markup language	Extended customization	Generated diagrams	Highlighting and linking of generated doc
	Windows	OS X	Javascript	HTML	PDF				
Sphinx	✓	✓	sphinx-js	✓	rst2pdf	.RST	10 themes; Python plugins	several in sphinx-contrib, e.g. using aafigure, actdiag, Google Chart, or gnuplot	Table of Contents, Index; cross referencing; syntax highlighting with Pygments
jsDOC	✓	✓	✓	✓	✗	M↓			
MkDocs	✓	✗	extension	✓	extension	M↓	built in themes: mkdocs and readthedocs; themes in the MkDocs wiki		
Doxygen	✓	✓	✗	✓	indirectly	M↓	With XSLT	dependency graphs, inheritance diagrams etc	
phpDocumentor	✓	✓	✗	✓	✓	.RST	Smarty-base templates	class inheritance diagrams	cross reference to generated documentation
pyDoc	✓	✗	✗	✓	✓	.RST	themes		

A comparison table of technical documentation tools evaluated

II.2.C. Pick your documentation tool

In the end, Sphinx was selected for the project, as it is a tool that helps create structured documentation. Using Sphinx, you can write your documentation using the restructured Text markup language. It also supports javascript, a language that Google App Script is based on. Sphinx then uses the Python docutils library to parse the restructured text files into various output formats such as HTML, PDFs, etc. For this project, all the technical documentation generated by using Sphinx was hosted on Github.

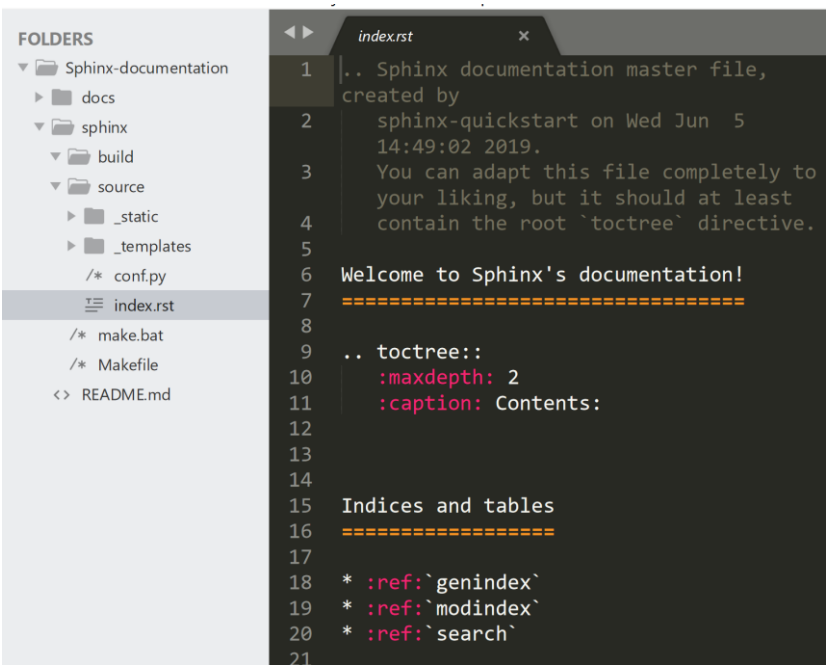
III. Setting up the selected tool: the example of Sphinx on GitHub

After the selection of the documentation tool, we had to set it up. Luckily, setting up Sphinx on GitHub was relatively simple. Here below are step-by-step instructions on how to do so on Windows Operating Systems:

1. Create your documentation repository on GitHub with proper description and clone it to your local computer
2. Open the cloned repository and create:
   2.1. a **docs** directory (where the materials from the web-site will go)
   2.2. a **sphinx** directory (where we are going to generate all the sphinx documentation)

3. Install Sphinx from PyPI (*a package management system used to install and manage software packages written in Python*)
   3.1. Install or upgrade pip by running the command ***pip install --upgrade pip*** in command prompt
   3.2. Install Sphinx by running the command ***pip install sphinx***
   3.3. Define document structure by switching to sphinx folder (Step 2.2) in command prompt and running the command ***sphinx-quickstart***

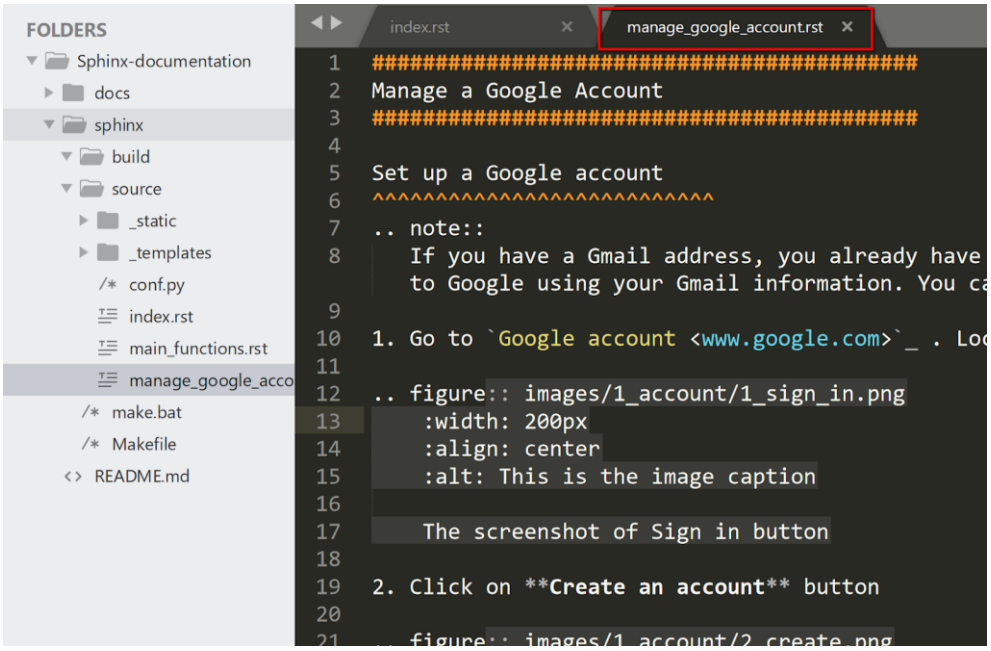
This will prompt you for information about your project and you can leave most of the settings as default. This will create a set of files, include the **index.rst** file, which is the master file for your documentation, and the **conf.py** with the most useful configuration values from a few questions it asks you. In **conf.py** you can add an additional extension, specify the link to the source code and, much more:



A screenshot of index.rst file

The index.rst file contains the root of the '**table of contents tree**' (or toctree): it is the central element to gather all documents into the documentation

4. Create the files and add content by using reStructuredText. Be careful, the file needs to be listed in the toctree (index.rst)



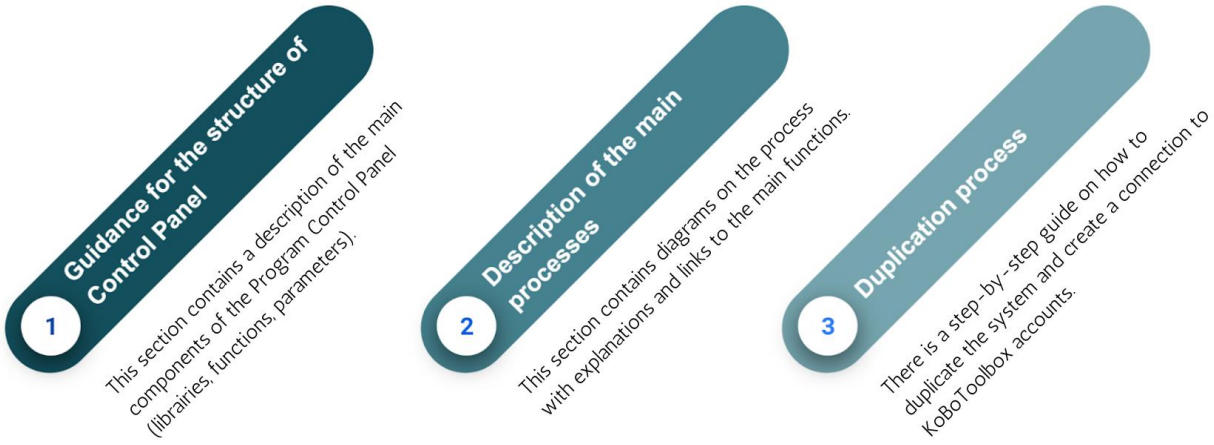
A screenshot of adding a new file 'manage_google_account'

5. Run the command ***make*** and specify ***html*** as output. This output can be used directly as a website as it has everything generated, including JavaScript and CSS files (Path is **sphinx/build/html**)

6. Copy all the files from **sphinx/build/html** to doc folder and host it on GitHub pages (Settings > GitHub Pages)

IV. Adding the system's content

Finally, once the project structure was in place, it was time for us to start creating the technical documentation.



A screenshot of the structure of the technical documentation of the project

Sphinx is our current tool-of-choice for documenting code, but it has lots of other use cases. You can use it for creating user guides, release notes, online help systems, adding information about your organization, policies and anything else that will be useful for your colleagues.