Using Jupyter Notebooks, MyST Markdown, & Sphinx to Publish Professional Quality HTML & LaTeX Documents

N. L. Dentinger-Anderson, C. L. Dentinger-Valentine

Jan 08, 2024

TABLE OF CONTENTS:

CHAPTER

ONE

THE GOAL: A PUBLISHING ENVIRONMENT FOR STATIC HTML & LATEX PDF

The purpose of this project is to streamline the publishing process for professional-quality scientific reports, books, and documentation. For these purposes, we will be using Jupyter Notebook, MyST Markdown, and Sphinx Documentation. These tools will serve as the backbone to our metadata-rich content, bringing depth and accessibility to the final products (a static HTML site and accompanying PDF book).

Whether you are just hearing of these tools or are looking to sharpen some skills, this tutorial will do its best to thoroughly improve your scientific deliverables.

Let's get a sense of scope before we drill in.

Objectives

There are many facets to this project, in this chapter, we will only concern ourselves with the big picture. Our main objectives by the end of this project will be to:

- Publish Jupyter Notebooks using MyST Markdown and Sphinx Documentation as tools.
- Build a reproducible python environment that is fully configured with metadata and extensive documentation.
- Implement custom design elements such as admonitions, support for emojis, widgets, and interactive content.
- Streamline the pipeline to as few commands as possible for continuous deployment.

1.1 The Results

By the conclusion of this tutorial you will be able to build and replicate an environment capable of publishing beautiful static websites and professional PDFLaTeX-rendered documents. Dozens of built-in HTML themes and granular control over the LaTeX formatting make for an endless sandbox of options but the stock design tools produce elegant results by themselves.

1.1.1 Gallery

Check out the gallery below to see how the previous version of this tutorial rendered.

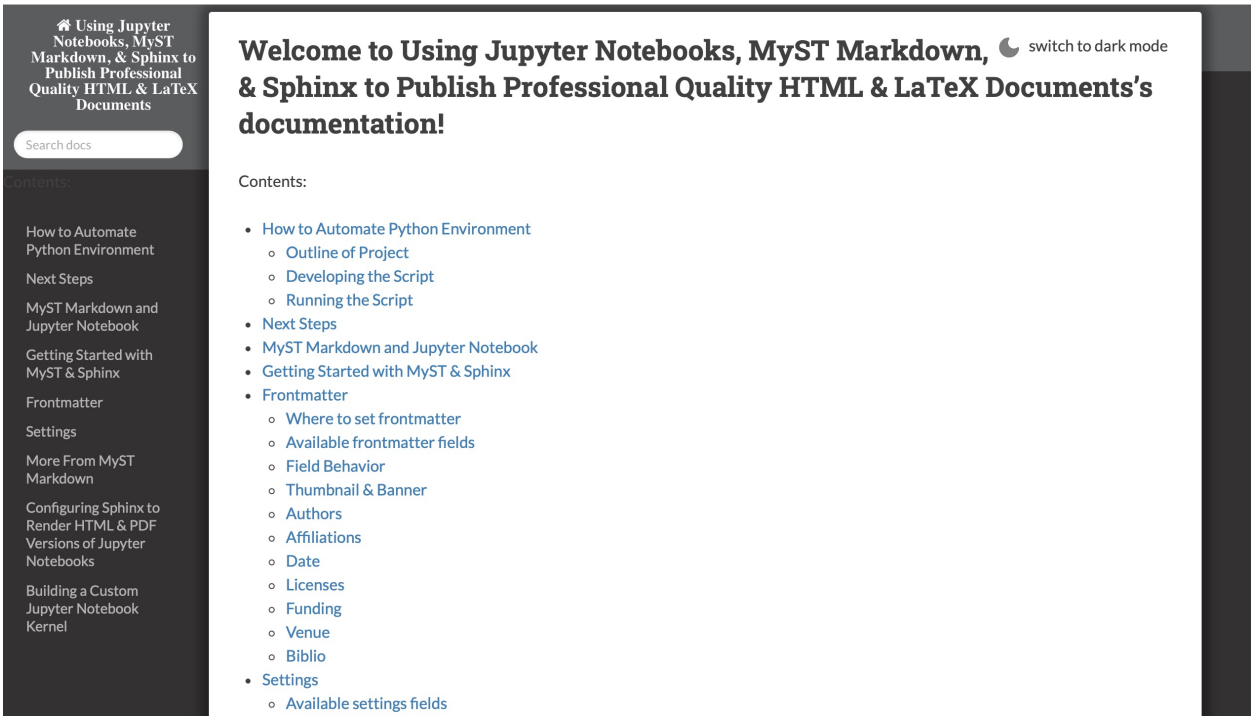


Fig. 1: *HTML renders like no other with adaptive UI built-in and preset themes to round out the breathability of the content.*

1.1.2 Contents of Build

1.1.2.1 Folder Contents

The basic structure of the final build directory is as follows:

- `docs` : This is the content folder and contains several other folders that make up the content that is to be converted and the content that is delivered.

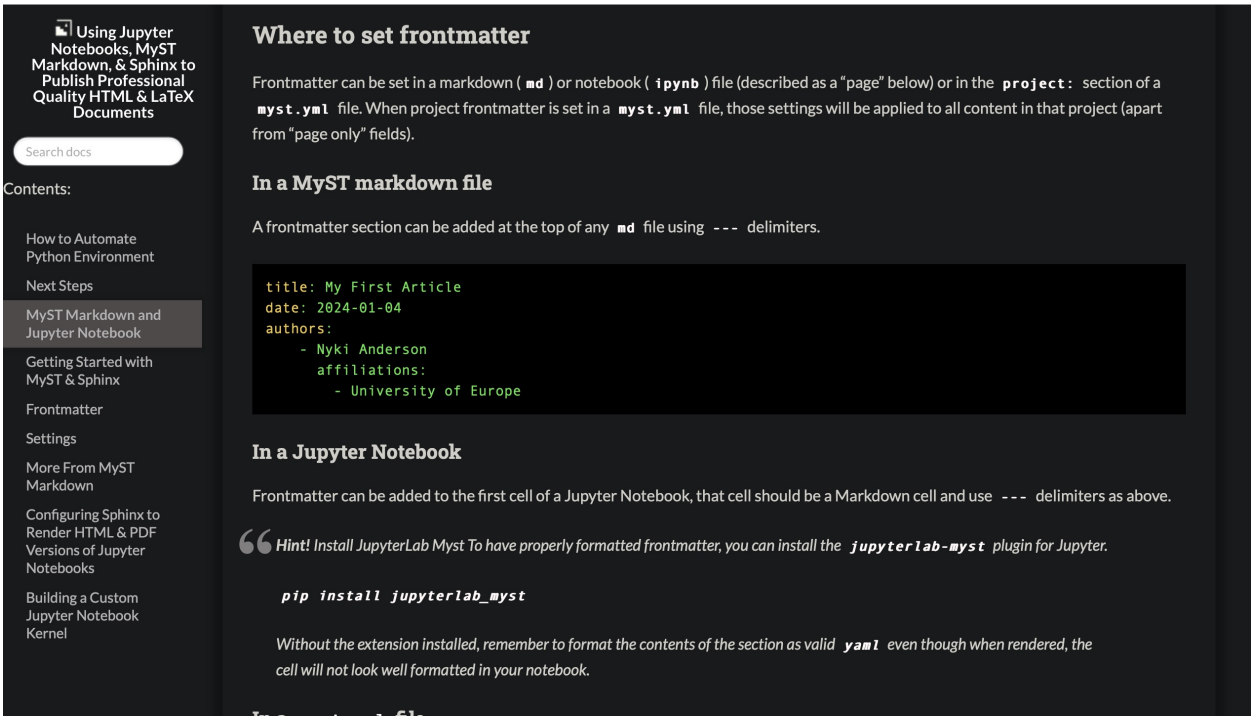


Fig. 2: *Sphinx even implements customizable syntax-highlighting thanks to its integration with Pygment themes.*

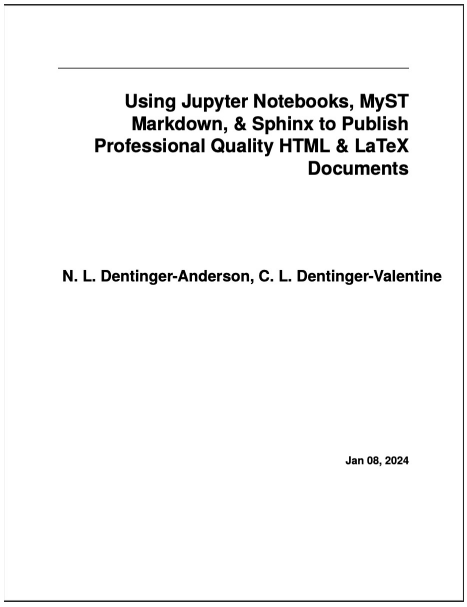
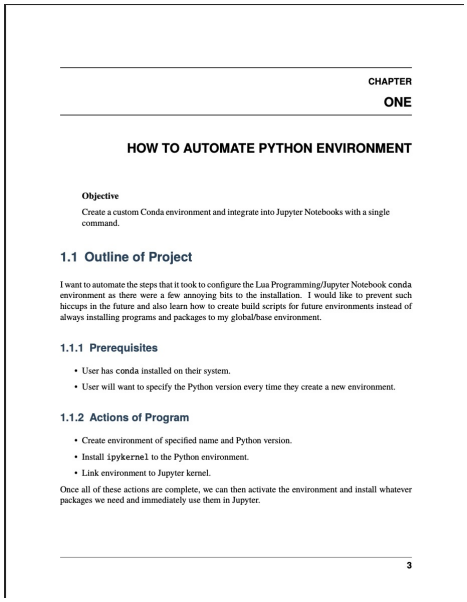


Fig. 3: *There's so much more to word-processing than font size and alignment. Sphinx and LaTeX work wonders together.*



CHAPTER

ONE

HOW TO AUTOMATE PYTHON ENVIRONMENT

Objective

Create a custom Conda environment and integrate into Jupyter Notebooks with a single command.

1.1 Outline of Project

I want to automate the steps that it took to configure the Lua Programming/Jupyter Notebook conda environment as there were a few annoying bits to the installation. I would like to prevent such hiccups in the future and also learn how to create build scripts for future environments instead of always installing programs and packages to my global/base environment.

1.1.1 Prerequisites

- User has conda installed on their system.
- User will want to specify the Python version every time they create a new environment.

1.1.2 Actions of Program

- Create environment of specified name and Python version.
- Install `ipykernel` to the Python environment.
- Link environment to Jupyter kernel.

Once all of these actions are complete, we can then activate the environment and install whatever packages we need and immediately use them in Jupyter.

3

Fig. 4: *The look and feel is organized with carefully auto-numbered sections and deliberate line breaking.*

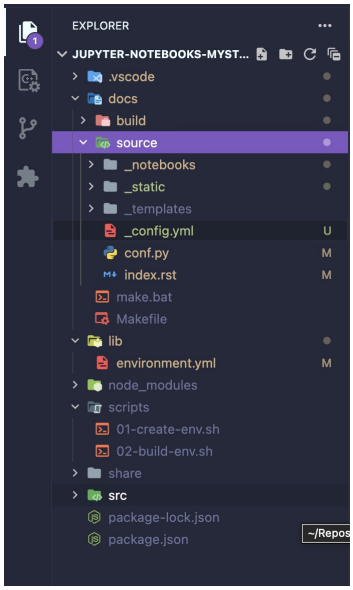


Fig. 5: *The final directory structure will look something like this when we've finished. More details on the exactly what each folder contains later.*

  - `build` : Inside this folder is where you will find either the LaTeX output or the HTML scripts.
  - `source` : Probably th most important folder, this is where content is kept and configuration `.yml`s housed.
    * `_notebooks` : You may have guessed it but this is our content folder. Any `.md`, `.rst`, or `.pynb` files that need to be rendered by Sphinx should be kept in here.
    * `_static` : Static assets like images and `.js` are stored here for easy reference by the content.
    * `_templates` : Any template files should be kept in this folder, though we don't make use of any throughout this tutorial.
    * `_config` : This file could be replaced with a `myst.yml` as they contain the same key:value pairs but I have yet to decide how I want to handle this.
    * `conf.py` : The most important configuration file, it contains all the directives for Sphinx and its integrations with the other apps. All extensions, file inclusions/exclusions, formatting, etc. are defined within.
    * `index.rst` : This is the **main** configuration file as it determines what is captured by the rendering process in a structure called the `toctree`.
  - `Makefile` and `make.bat` :

- `lib` : This contains any dependencies and packages that are needed for the environment including an `environment.yml` file with all the packages installed by the virtual `conda` environment that controls the whole build.
- `node_modules` : Contains the build packages for `node.js` though we don't make use of them in this tutorial.
- `scripts` : Two `bash` scripts are necessary to bring everything together with one command. (More on these soon.)
- `share` : This is where we store the files that derive any custom kernels we build or install.
- `src` : The interpreters for the kernels are kept here.
- `package.json` and `package-lock.json` : These two files maintain the versions of all node packages and describe the project.

1.2 The Road Ahead

This is a pretty massive undertaking when I think about the path that has led me to this point in the project. This project started out as research for another project that involved lua programming language and Pandoc exporting. Through much trial and error, I can say that I prefer this solution though it does not directly apply to the application I had before which was to export my Obsidian vault files to PDF elegantly (though it seems within the realm of possibility).

Still, it has been massively rewarding to go through this process and I hope by sharing it with you, it can make an impact on your productivity too. Here's a look ahead.

1.2.1 The Environment: A Local conda Installation

We will utilize Anaconda or Miniconda package manager conda for the majority of our environment build. We will of course automate the installation of packages though I will mention a few significant ones more critically. Automation will come in the form of very minimal bash scripting –though I must mention it was much more involved when I was building the xeus-lua kernel from source.

As new packages are found necessary, the running version control file, lib/environment.yml will need to be updated accordingly. Keeping this running "log" of what is being installed is what makes this project so valuable in the long run because you are going to want a portable installation that can adapt to whatever your next publication need is.

1.2.2 The Configuration: Integrating Three Tools into One

This will, by far, be the most time consuming section to cover, as it took me days to come to the correct solution for integrating all three tools. There is extensive documentation –as you might guess– for all three of these programs but there are significant holes in communication as far as where configuration directives can legally be called and exactly where to put the files when you figure it out.

Folding in all three tools was a challenge, made all the harder by the fact that I opted to keep the installations local rather than use something like JupyterLab which actually has MyST Markdown preview capabilities. I stuck with that decision because I wanted to be able to control exactly how much unfinished content was "out there" on a server. I imagine using these tools to publish and finalize rather proprietary information at some point and I wanted to know how to do it in house before considering security implications.

Staying Local

Staying local also meant I got to keep using VS Code which runs Jupyter Notebook natively and does the best job out of any IDE I've ever used at maintaining a cohesive workspace. Not to mention the ability to use the command line right within the application itself. And while most of the VS

Code extensions cannot reach the Jupyter kernel directly, most of the perks of the Jupyter Notebook UI are available through the command prompt.

As far as the configuration files go themselves, they are written in `.yml` mostly with a few exceptions. The Sphinx configuration file is a python script called conf.py, the *index.rst* is written in reStructureText (a derivative of Markdown), and there are a couple versions of a Makefile –or-make.bat– to bring it all together. Details are best left to that respective chapter.

1.2.3 The Deployment: Publishing Our Work in its Finest Format

This was probably the most frustrating part of the whole process because it was dependent on a lot of tools and techniques that I've never used before. The greatest difficulty came in the form of learning GitHub Actions and Workflows –or rather, not learning– how to deploy to GitHub Pages in that way. I also experimented with Read the Docs deployment which is still tied to a GitHub repository but carries its own documentation and workflows. I found this method much less transparent than the pure GitHub method and eventually found the simplest tutorial ever. In this chapter, we will explore the various methods and how they are supposed to work.

There is also some value in refining the `PDFLatex` workflow so that it is more customized to the document itself rather than full of assumptions and errors. I find that this method of publishing is more in the wind than the HTML export and that's because LaTeX is programmatically structured rather without the benefits of the linear progression and infinite scroll of a website. Still, the approach as it stands currently has quite the payoff and is something I am excited to share.

1.3 Project Expansion

This project could see a lot more content as an addendum in the future. I see many ways to improve the pipeline and certainly more ways to get value out of the capabilities of all three tools. Jupyter Notebooks intrigues me as a likely replacement –or at least equally tempting option– for use as my go to note-taking app. I love Obsidian more than most people love their children and have made tremendous growth in terms of Plugin integration and authorship but it doesn't seem to be as refined or elegant as Jupyter.

Further, as I start to embark on legitimate research that I might wish to share with the world, I can tell that Sphinx will be a long time friend. I do enjoy making tutorials about the things I learn so publishing in this format will certainly be a step up from the dreaded GitHub `docs` repositories I've come accustomed to submitting.

This project could evolve even before it is done being written as there were many loose ends that I just had to move away from because they were too far outside the scope of the immediate project. But I would potentially like to address them within the same tutorial as addenda. These topics include:

- Developing a custom Jupyter kernel with the capability of running both `xeus-lua` and `shell` commands or at least the potential to make use of `IPython` magic commands.
- Building a full-service GitHub repo that can be installed using a simple `Makefile` or `bash` script. The project is close on delivering this kind of universality but I am not quite sure how much further I would need to take it, so more research is required.
- Integrating a more flexible `node.js` kernel capable of running insecure web applications from within the Jupyter Notebook with a tool like OWASP-ZAP for performing penetration testing. I found a couple of tutorials on this topic and it is certainly on my radar as a possible alternative to using Docker for the same purposes.
- Cleaning up the repository and environment install so that it is leaner and faster.
- Implement custom admonitions with graphic much like the native-MyST Markdown flavored admonitions.

1.4 Closing Remarks

It is still uncertain how many, if any of these topics I will be able to cover in this tutorial before another project takes precedent. But if any of these sound more useful than others, do not hesitate to reach out in the comments of this repo.

Now, let's get our hands on some code!

CHAPTER

TWO

THE ENVIRONMENT: A LOCAL `CONDA` INSTALLATION

Now that we can start in with the actual coding, let's get our ducks in a row. This chapter will focus on installing, building, and fine-tuning our development environment, all of which will run on a `Python` virtual environment. This will be done locally for a number of reasons which may all become clear later in the *Why A Virtual Environment?*. All of this will allow us to use a `python3.12` kernel for use within the Jupyter Notebook interface.

Objectives

The aim of this chapter is to:

- Fully characterize the set of packages, tools, and other considerations that are required for Jupyter Notebook to function properly throughout its use.
- Determine what (if any) limitations we will have when developing a build-time script for a local `conda` virtual environment vs. installing our full build into the `base` environment.
- Automate the build process with a single command and test that environment is equivalent to the one we develop initially.

2.1 Prerequisites

The setup is fairly specific to the exact tools and their versions, though I did not have any issues that required me to downgrade any packages. Though, keep this detail in mind throughout the project, as any deviations could potentially break the build.

We will be using VS Code's local version of the Jupyter Notebook software but little configuration is necessary on that front.

2.1.1 Why A Virtual Environment?

2.1.2 Considering Versions

CHAPTER

THREE

THE CONFIGURATION: INTEGRATING THREE TOOLS INTO ONE

CHAPTER

FOUR

THE DEPLOYMENT: PUBLISHING OUR WORK IN ITS FINEST FORMAT

The purpose of this whole venture was to produce professional quality deliverables that let the content speak for itself. In this chapter, we will introduce a few of the options for doing so –for free– including publishing our static documentation site on GitHub Pages via three methods: (1) using the `ghp-import` command at the command line (which was the only way I got working so far), (2) Actions and Workflows, and (3) from a branch. As well as a third method which involves publishing through your GitHub repository to the Read the Docs client.

Objectives

This chapter is due to wrap up the main purpose of the project, which was to share our documentation and ideas with a greater audience. To that aim, we will:

- Finalize any content we wish to publish and determine the format(s) we are confident will reach the broadest audience or satisfy whatever deliverables we might have.
- Produce both a collection of linkable static web scripts as well as a LaTeX-rendered PDF printout worthy of professional scrutiny.
- Determine which methods of distribution work, which are more difficult or situation-biased, and which fit best with our current workflow.
- Clear up any remaining configuration errors and answer lingering examples of broken code or functionality.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search