nbsphinx

Release "0.4.3+ds (Debian 0.4.3+ds-1)"

Matthias Geier

"2019-10-05"

Contents

nbsphinx is a Sphinx[1] extension that provides a source parser for *.ipynb files. Custom Sphinx directives are used to show Jupyter Notebook[2] code cells (and of course their results) in both HTML and LaTeX output. Un-evaluated notebooks – i.e. notebooks without stored output cells – will be automatically executed during the Sphinx build process.

Quick Start:

1. Install nbsphinx
2. Edit your conf.py and add 'nbsphinx' to extensions.
3. Edit your index.rst and add the names of your *.ipynb files to the toctree.
4. Run Sphinx!

Online documentation (and example of use): http://nbsphinx.readthedocs.io/

Source code repository (and issue tracker): https://github.com/spatialaudio/nbsphinx/

License: MIT – see the file LICENSE for details.

All content shown below – except for the section *Normal reStructuredText Files* (page 37) – was generated from Jupyter notebooks.

The following section was generated from doc/installation.ipynb

1 Installation

Note that some packages may be out of date. You can always get the newest nbsphinx release from PyPI[3] (using pip). If you want to try the latest development version, have a look at the file CONTRIBUTING.rst[4].

1.1 nbsphinx Packages

[5]

If you are using the conda package manager (e.g. with Anaconda[6] for Linux/macOS/Windows), you can install nbsphinx from the conda-forge[7] channel:

```
conda install -c conda-forge nbsphinx
```

If you are using Linux, there are packages available for many distributions.

[8]

[9]

On any platform, you can also install nbsphinx with pip, Python's own package manager:

[1] https://www.sphinx-doc.org/
[2] https://jupyter.org/
[3] https://pypi.org/project/nbsphinx
[4] https://github.com/spatialaudio/nbsphinx/blob/master/CONTRIBUTING.rst
[5] https://anaconda.org/conda-forge/nbsphinx
[6] https://www.anaconda.com/distribution/
[7] https://conda-forge.org/
[8] https://repology.org/project/python:nbsphinx/versions
[9] https://pypi.org/project/nbsphinx

```
python3 -m pip install nbsphinx --user
```

If you want to install it system-wide for all users (assuming you have the necessary rights), just drop the --user flag.

To upgrade an existing nbsphinx installation to the newest release, use the --upgrade flag:

```
python3 -m pip install nbsphinx --upgrade --user
```

If you suddenly change your mind, you can un-install it with:

```
python3 -m pip uninstall nbsphinx
```

Depending on your Python installation, you may have to use python instead of python3.

1.2 nbsphinx Prerequisites

Some of the aforementioned packages will install some of these prerequisites automatically, some of the things may be already installed on your computer anyway.

1.2.1 Python

Of course you'll need Python, because both Sphinx and nbsphinx are implemented in Python. There are many ways to get Python. If you don't know which one is best for you, you can try Anaconda[10].

1.2.2 Sphinx

You'll need Sphinx[11] as well, because nbsphinx is just a Sphinx extension and doesn't do anything on its own.

If you use conda, you can get Sphinx from the conda-forge channel[12]:

```
conda install -c conda-forge sphinx
```

Alternatively, you can install it with pip (see below):

```
python3 -m pip install Sphinx --user
```

1.2.3 pip

Recent versions of Python already come with pip pre-installed. If you don't have it, you can install it manually[13].

1.2.4 pandoc

The stand-alone program pandoc[14] is used to convert Markdown content to something Sphinx can understand. You have to install this program separately, ideally with your package manager. If you are using conda, you can install pandoc from the conda-forge channel[15]:

[10] https://www.anaconda.com/distribution/
[11] https://www.sphinx-doc.org/
[12] https://anaconda.org/conda-forge/sphinx
[13] https://pip.pypa.io/en/latest/installing/
[14] https://pandoc.org/
[15] https://anaconda.org/conda-forge/pandoc

```
conda install -c conda-forge pandoc
```

If that doesn't work out for you, have a look at pandoc's installation instructions[16].

Note:

The use of pandoc in nbsphinx is temporary, but will likely stay that way for a long time, see issue #36[17].

1.2.5 Pygments Lexer for Syntax Highlighting

To get proper syntax highlighting in code cells, you'll need an appropriate *Pygments lexer*. This of course depends on the programming language of your Jupyter notebooks (more specifically, the pygments_lexer metadata of your notebooks).

For example, if you use Python in your notebooks, you'll have to have the IPython package installed, e.g. with

```
conda install -c conda-forge ipython
```

or

```
python3 -m pip install IPython --user
```

Note:

If you are using Anaconda with the default channel and syntax highlighting in code cells doesn't seem to work, you can try to install IPython from the conda-forge channel or directly with pip, or as a work-around, add 'IPython.sphinxext.ipython_console_highlighting' to extensions in your conf.py.

For details, see Anaconda issue #1430[18] and nbsphinx issue #24[19].

1.2.6 Jupyter Kernel

If you want to execute your notebooks during the Sphinx build process (see *Controlling Notebook Execution* (page 28)), you need an appropriate Jupyter kernel[20] installed.

For example, if you use Python, you should install the ipykernel package, e.g. with

```
conda install -c conda-forge ipykernel
```

or

```
python3 -m pip install ipykernel --user
```

If you created your notebooks yourself with Jupyter, it's very likely that you have the right kernel installed already.

doc/installation.ipynb ends here.

[16] https://pandoc.org/installing.html
[17] https://github.com/spatialaudio/nbsphinx/issues/36
[18] https://github.com/ContinuumIO/anaconda-issues/issues/1430
[19] https://github.com/spatialaudio/nbsphinx/issues/24
[20] https://jupyter.readthedocs.io/en/latest/projects/kernels.html

2 Usage

2.1 Sphinx Setup

In the directory with your notebook files, run this command (assuming you have Sphinx[21] installed already):

```
python3 -m sphinx.cmd.quickstart
```

Answer the questions that appear on the screen. In case of doubt, just press the <Return> key repeatedly to take the default values.

After that, there will be a few brand-new files in the current directory. You'll have to make a few changes to the file named conf.py. You should at least check if those two variables contain the right things:

```
extensions = [
    'nbsphinx',
    'sphinx.ext.mathjax',
]
exclude_patterns = ['_build', '**.ipynb_checkpoints']
```

For an example, see this project's conf.py file.

Once your conf.py is in place, edit the file named index.rst and add the file names of your notebooks (without the .ipynb extension) to the toctree[22] directive. For an example, see this project's doc/index.rst file.

2.2 Running Sphinx

To create the HTML pages, use this command:

```
python3 -m sphinx <source-dir> <build-dir>
```

If you have many notebooks, you can do a parallel build by using the -j option:

```
python3 -m sphinx <source-dir> <build-dir> -j<number-of-processes>
```

For example, if your source files are in the current directory and you have 4 CPU cores, you can run this:

```
python3 -m sphinx . _build -j4
```

Afterwards, you can find the main HTML file in _build/index.html.

Subsequent builds will be faster, because only those source files which have changed will be re-built. To force re-building all source files, use the -E option.

Note:

By default, notebooks will be executed during the Sphinx build process only if they do not have any output cells stored. See *Controlling Notebook Execution* (page 28).

To create LaTeX output, use:

[21] https://www.sphinx-doc.org/
[22] https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-toctree

```
python3 -m sphinx <source-dir> <build-dir> -b latex
```

If you don't know how to create a PDF file from the LaTeX output, you should have a look at Latexmk[23] (see also this tutorial[24]).

Sphinx can automatically check if the links you are using are still valid. Just invoke it like this:

```
python3 -m sphinx <source-dir> <build-dir> -b linkcheck
```

2.3 Watching for Changes with sphinx-autobuild

If you think it's tedious to run the Sphinx build command again and again while you make changes to your notebooks, you'll be happy to hear that there is a way to avoid that: sphinx-autobuild[25]!

It can be installed with

```
python3 -m pip install sphinx-autobuild --user
```

You can start auto-building your files with

```
python3 -m sphinx_autobuild <source-dir> <build-dir>
```

This will start a local webserver which will serve the generated HTML pages at http://localhost:8000/. Whenever you save changes in one of your notebooks, the appropriate HTML page(s) will be re-built and when finished, your browser view will be refreshed automagically. Neat!

You can also abuse this to auto-build the LaTeX output:

```
python3 -m sphinx_autobuild <source-dir> <build-dir> -b latex
```

However, to auto-build the final PDF file as well, you'll need an additional tool. Again, you can use latexmk for this (see *above* (page 6)). Change to the build directory and run

```
latexmk -pdf -pvc
```

If your PDF viewer isn't opened because of LaTeX build errors, you can use the command line flag -f to *force* creating a PDF file.

2.4 Automatic Creation of HTML and PDF output on readthedocs.org

There are two different methods, both of which are described below.

In both cases, you'll first have to create an account on https://readthedocs.org/ and connect your GitLab/Github/Bitbucket account. Instead of connecting, you can also manually add any publicly available Git/Subversion/Mercurial/Bazaar repository.

After doing the steps described below, you only have to "push" to your repository, and the HTML pages and the PDF file of your stuff are automagically created on readthedocs.org. Awesome!

You can even have different versions of your stuff, just use Git tags and branches and select in the readthedocs.org settings (under "Admin", "Versions") which of those should be created.

Note:

[23] http://personal.psu.edu/jcc8//software/latexmk-jcc/
[24] https://mg.readthedocs.io/latexmk.html
[25] https://pypi.org/project/sphinx-autobuild

If you want to execute notebooks (see *Controlling Notebook Execution* (page 28)), you'll need to install the appropriate Jupyter kernel. In the examples below, the IPython kernel ist installed from the packet ipykernel.

2.4.1 Using requirements.txt

1. Create a file named requirements.txt (or whatever name you wish) in your repository containing the required pip packages:

   ```
   ipykernel
   nbsphinx
   ```

   You can also install directly from Github et al., using a specific branch/tag/commit, e.g.

   ```
   git+https://github.com/spatialaudio/nbsphinx.git@master
   ```

2. In the "Advanced Settings" on readthedocs.org, specify the path to your requirements.txt file (or however you called it) in the box labeled "Requirements file". Kinda obvious, isn't it?

3. Still in the "Advanced Settings", make sure the right Python interpreter is chosen. This must be the same version (2.x or 3.x) as you were using in your notebooks!

2.4.2 Using conda

1. Create a file named readthedocs.yml in the main directory of your repository, containing the name of yet another file:

   ```
   conda:
     file: readthedocs-environment.yml
   ```

2. Create the file mentioned above. You can choose whatever name you want (it may also live in a subdirectory, e.g. doc/environment.yml), it just has to match whatever is specified in readthedocs.yml. The second file describes a conda environment[26] and should contain something like this:

   ```
   channels:
     - conda-forge
   dependencies:
     - python>=3
     - pandoc
     - ipykernel
     - pip:
       - nbsphinx
   ```

   It is up to you if you want to install nbsphinx with conda or with pip (but note that the conda package might be outdated). And you can of course add further conda and pip packages. You can also install packages directly from Github et al., using a specific branch/tag/commit, e.g.

   ```
   - pip:
     - git+https://github.com/spatialaudio/nbsphinx.git@master
   ```

Note:

The specification of the conda-forge channel is recommended because it tends to have more recent package versions than the default channel.

[26] https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

Note:

Most of the "Advanced Settings" on readthedocs.org will be ignored if you have a `readthedocs.yml` file. In this file you can control all the settings, see https://docs.readthedocs.io/en/latest/yaml-config.html.

Warning:

If you have a very long repository name (or branch name), you might run into this quite obscure problem: 'placeholder too short'[27].

2.5 HTML Themes

The `nbsphinx` extension does *not* provide its own theme, you can use any of the available themes or create a custom one[28], if you feel like it.

The following (incomplete) list of themes contains up to three links for each theme:

1. The documentation (or the official sample page) of this theme (if available; see also the documentation of the built-in Sphinx themes[29])
2. How the `nbsphinx` documentation looks when using this theme
3. How to enable this theme using either `requirements.txt` or `readthedocs.yml` and theme-specific settings (in some cases)

2.5.1 Sphinx's Built-In Themes

- agogo: example[30], usage[31]
- alabaster[32]: example[33], usage[34]
- bizstyle: example[35], usage[36]
- classic: example[37], usage[38]
- haiku: example[39], usage[40]
- nature: example[41], usage[42]
- pyramid: example[43], usage[44]

[27] https://github.com/readthedocs/readthedocs.org/issues/1902
[28] https://www.sphinx-doc.org/en/master/theming.html#creating-themes
[29] https://www.sphinx-doc.org/en/master/usage/theming.html#builtin-themes
[30] https://nbsphinx.readthedocs.io/en/agogo-theme/
[31] https://github.com/spatialaudio/nbsphinx/compare/agogo-theme%5E...agogo-theme
[32] https://alabaster.readthedocs.io/
[33] https://nbsphinx.readthedocs.io/en/alabaster-theme/
[34] https://github.com/spatialaudio/nbsphinx/compare/alabaster-theme%5E...alabaster-theme
[35] https://nbsphinx.readthedocs.io/en/bizstyle-theme/
[36] https://github.com/spatialaudio/nbsphinx/compare/bizstyle-theme%5E...bizstyle-theme
[37] https://nbsphinx.readthedocs.io/en/classic-theme/
[38] https://github.com/spatialaudio/nbsphinx/compare/classic-theme%5E...classic-theme
[39] https://nbsphinx.readthedocs.io/en/haiku-theme/
[40] https://github.com/spatialaudio/nbsphinx/compare/haiku-theme%5E...haiku-theme
[41] https://nbsphinx.readthedocs.io/en/nature-theme/
[42] https://github.com/spatialaudio/nbsphinx/compare/nature-theme%5E...nature-theme
[43] https://nbsphinx.readthedocs.io/en/pyramid-theme/
[44] https://github.com/spatialaudio/nbsphinx/compare/pyramid-theme%5E...pyramid-theme

- scrolls: example[45], usage[46]
- traditional: example[47], usage[48]

2.5.2 3rd-Party Themes

- basicstrap[49]: example[50], usage[51]
- better[52]: example[53], usage[54]
- bootstrap[55]: example[56], usage[57]
- cloud[58]: example[59], usage[60]
- dotted[61]: example[62], usage[63]
- guzzle_sphinx_theme[64]: example[65], usage[66]
- julia[67]: example[68], usage[69]
- jupyter[70]: example[71], usage[72]
- sizzle_sphinx_theme[73]: example[74], usage[75]
- sphinx_py3doc_enhanced_theme[76]: example[77], usage[78]
- sphinx_rtd_theme[79]: example[80], usage[81]

If you know of another Sphinx theme that should be included here, please open an issue on Github[82]. An overview of many more themes can be found at https://sphinx-themes.org/.

[45] https://nbsphinx.readthedocs.io/en/scrolls-theme/
[46] https://github.com/spatialaudio/nbsphinx/compare/scrolls-theme%5E...scrolls-theme
[47] https://nbsphinx.readthedocs.io/en/traditional-theme/
[48] https://github.com/spatialaudio/nbsphinx/compare/traditional-theme%5E...traditional-theme
[49] https://pythonhosted.org/sphinxjp.themes.basicstrap/
[50] https://nbsphinx.readthedocs.io/en/basicstrap-theme/
[51] https://github.com/spatialaudio/nbsphinx/compare/basicstrap-theme%5E...basicstrap-theme
[52] https://sphinx-better-theme.readthedocs.io/
[53] https://nbsphinx.readthedocs.io/en/better-theme/
[54] https://github.com/spatialaudio/nbsphinx/compare/better-theme%5E...better-theme
[55] https://sphinx-bootstrap-theme.readthedocs.io/
[56] https://nbsphinx.readthedocs.io/en/bootstrap-theme/
[57] https://github.com/spatialaudio/nbsphinx/compare/bootstrap-theme%5E...bootstrap-theme
[58] https://pythonhosted.org/cloud_sptheme/
[59] https://nbsphinx.readthedocs.io/en/cloud-theme/
[60] https://github.com/spatialaudio/nbsphinx/compare/cloud-theme%5E...cloud-theme
[61] https://pythonhosted.org/sphinxjp.themes.dotted/
[62] https://nbsphinx.readthedocs.io/en/dotted-theme/
[63] https://github.com/spatialaudio/nbsphinx/compare/dotted-theme%5E...dotted-theme
[64] https://github.com/guzzle/guzzle_sphinx_theme
[65] https://nbsphinx.readthedocs.io/en/guzzle-theme/
[66] https://github.com/spatialaudio/nbsphinx/compare/guzzle-theme%5E...guzzle-theme
[67] https://github.com/JuliaLang/JuliaDoc
[68] https://nbsphinx.readthedocs.io/en/julia-theme/
[69] https://github.com/spatialaudio/nbsphinx/compare/julia-theme%5E...julia-theme
[70] https://github.com/jupyter/jupyter-sphinx-theme/
[71] https://nbsphinx.readthedocs.io/en/jupyter-theme/
[72] https://github.com/spatialaudio/nbsphinx/compare/jupyter-theme%5E...jupyter-theme
[73] https://docs.red-dove.com/sphinx_sizzle_theme/
[74] https://nbsphinx.readthedocs.io/en/sizzle-theme/
[75] https://github.com/spatialaudio/nbsphinx/compare/sizzle-theme%5E...sizzle-theme
[76] https://github.com/ionelmc/sphinx-py3doc-enhanced-theme
[77] https://nbsphinx.readthedocs.io/en/py3doc-enhanced-theme/
[78] https://github.com/spatialaudio/nbsphinx/compare/py3doc-enhanced-theme%5E...py3doc-enhanced-theme
[79] https://github.com/readthedocs/sphinx_rtd_theme
[80] https://nbsphinx.readthedocs.io/en/rtd-theme/
[81] https://github.com/spatialaudio/nbsphinx/compare/rtd-theme%5E...rtd-theme
[82] https://github.com/spatialaudio/nbsphinx/issues

2.6 Using Notebooks with Git

Git[83] is extremely useful for managing source code and it can and should also be used for managing Jupyter notebooks. There is one caveat, however: Notebooks can contain output cells with rich media like images, plots, sounds, HTML, JavaScript and many other types of bulky machine-created content. This can make it hard to work with Git efficiently, because changes in those bulky contents can completely obscure the more interesting human-made changes in text and source code. Working with multiple collaborators on a notebook can become very tedious because of this.

It is therefore highly recommended that you remove all outputs from your notebooks before committing changes to a Git repository (except for the reasons mentioned in *Pre-Executing Notebooks* (page 28)).

If there are no output cells in a notebook, `nbsphinx` will by default execute the notebook, and the pages generated by Sphinx will therefore contain all the output cells. See *Controlling Notebook Execution* (page 28) for how this behavior can be customized.

In the Jupyter Notebook application, you can manually clear all outputs by selecting "Cell" → "All Output" → "Clear" from the menu. In JupyterLab, the menu items are "Edit" → "Clear All Outputs".

There are several tools available to remove outputs from multiple files at once without having to open them separately. You can even include such a tool as "clean/smudge filters" into your Git workflow, which will strip the output cells automatically whenever a Git command is executed. For details, have a look at those links:

- https://github.com/kynan/nbstripout
- https://github.com/toobaz/ipynb_output_filter
- https://tillahoffmann.github.io/2017/04/17/versioning-jupyter-notebooks-with-git.html
- http://timstaley.co.uk/posts/making-git-and-jupyter-notebooks-play-nice/
- https://pascalbugnion.net/blog/ipython-notebooks-and-git.html
- https://github.com/choldgraf/nbclean
- https://jamesfolberth.org/articles/2017/08/07/git-commit-hook-for-jupyter-notebooks/

doc/usage.ipynb ends here.

The following section was generated from doc/markdown-cells.ipynb

3 Markdown Cells

We can use *emphasis*, **boldface**, `preformatted text`.

> It looks like strike-out text is not supported: [STRIKEOUT:strikethrough].

- Red
- Green
- Blue

---

1. One
2. Two
3. Three

Arbitrary Unicode characters should be supported, e.g. łßō. Note, however, that this only works if your HTML browser and your LaTeX processor provide the appropriate fonts.

[83] https://git-scm.com/

3.1 Equations

Inline equations like $e^{i\pi} = -1$ can be created by putting a LaTeX expression between two dollar signs, like this: `$\text{e}^{i\pi} = -1$`.

Note:

Avoid leading and trailing spaces around math expressions, otherwise errors like the following will occur when Sphinx is running:

```
ERROR: Unknown interpreted text role "raw-latex".
```

See also the pandoc docs[84]:

> Anything between two $ characters will be treated as TeX math. The opening $ must have a non-space character immediately to its right, while the closing $ must have a non-space character immediately to its left, and must not be followed immediately by a digit.

Equations can also be displayed on their own line like this:

$$\int_{-\infty}^{\infty} f(x)\delta(x - x_0)dx = f(x_0). \tag{1}$$

This can be done by simply using one of the LaTeX math environments, like so:

```
\begin{equation}
\int\limits_{-\infty}^\infty f(x) \delta(x - x_0) dx = f(x_0)
\end{equation}
```

3.1.1 Automatic Equation Numbering

This is not automatically enabled in Jupyter notebooks, but you can install a notebook extension in order to enable equation numbering: https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/nbextensions/equation-numbering/readme.html.

Automatic Equation Numbering is enabled on https://nbviewer.jupyter.org/, see e.g. the latest version of this very notebook at the link https://nbviewer.jupyter.org/github/spatialaudio/nbsphinx/blob/master/doc/markdown-cells.ipynb.

When using `nbsphinx`, you can use the following `mathjax_config` setting in your `conf.py` file to enable automatic equation numbering in HTML output. In LaTeX output, the equations are numbered by default.

```
mathjax_config = {
    'TeX': {'equationNumbers': {'autoNumber': 'AMS', 'useLabelIds': True}},
}
```

You can use `\label{...}` to give a unique label to an equation:

$$\phi = \frac{1 + \sqrt{5}}{2} \tag{2}$$

[84] https://pandoc.org/MANUAL.html#math

```
\begin{equation}
\phi = \frac{1 + \sqrt{5}}{2}
\label{golden-mean}
\end{equation}
```

If automatic equation numbering is enabled, you can later reference that equation using its label. You can use \eqref{golden-mean} for a reference with parentheses: (2), or \ref{golden-mean} for a reference without them: 2.

In HTML output, these equation references only work for equations within a single HTML page. In LaTeX output, equations from other notebooks can be referenced, e.g. (08.15).

3.1.2 Manual Equation Numbering

If you prefer to assign equation numbers (or some kind of names) manually, you can do so with \tag{...}:

$$a^2 + b^2 = c^2 \tag{99.4}$$

```
\begin{equation}
a^2 + b^2 = c^2
\tag{99.4}
\label{pythagoras}
\end{equation}
```

The above equation has the number 99.4.

3.2 Citations

According to https://nbconvert.readthedocs.io/en/latest/latex_citations.html, nbconvert supports citations using a special HTML-based syntax. nbsphinx supports the same syntax.

Example: [KRKP+16].

```
<cite data-cite="kluyver2016jupyter">Kluyver et al. (2016)</cite>
```

You don't actually have to use <cite>, any inline HTML tag can be used, e.g. : [PGH11].

```
<strong data-cite="perez2011python">Python: An Ecosystem for Scientific Computing</strong>
```

You'll also have to define a list of references, see *the section about references* (page 39).

There is also a Notebook extension which may or may not be useful: https://github.com/takluyver/cite2c.

3.3 Code

We can also write code with nice syntax highlighting:

```
print("Hello, world!")
```

3.4 Tables

A	B	A and B
False	False	False
True	False	False
False	True	False
True	True	True

3.5 Images

Local image: 


```
![Jupyter notebook icon](images/notebook_icon.png)
```

Remote image:

```
![Python logo (remote)](https://www.python.org/static/img/python-logo-large.png)
```

3.5.1 SVG support for LaTeX

LaTeX doesn't support SVG images, but there are Sphinx extensions that can be used for automatically converting SVG images for inclusion in LaTeX output.

Just include one of the following options in the list of `extensions` in your `conf.py` file.

- `'sphinxcontrib.inkscapeconverter'` or `'sphinxcontrib.rsvgconverter'`: See https://github.com/missinglinkelectronics/sphinxcontrib-svg2pdfconverter for installation instructions.

  The external programs `inkscape` or `rsvg-convert` (Debian/Ubuntu package `librsvg2-bin`) are needed, respectively.

- `'sphinx.ext.imgconverter'`: This is a built-in Sphinx extension, see https://www.sphinx-doc.org/en/master/usage/extensions/imgconverter.html.

  This needs the external program `convert` from *ImageMagick*.

  The disadvantage of this extension is that SVGs are converted to bitmap images.

If one of those extensions is installed, SVG images can be used even for LaTeX output:

```
![Python logo](images/python_logo.svg)
```

Remote SVG images can also be used:

```
![Jupyter logo](https://jupyter.org/assets/main-logo.svg)
```

3.6 Cell Attachments

Images can also be embedded in the notebook itself. Just drag an image file into the Markdown cell you are just editing or copy and paste some image data from an image editor/viewer.

The generated Markdown code will look just like a "normal" image link, except that it will have an `attachment:` prefix:

```
![a stick figure](attachment:stickfigure.png)
```

This is a cell attachment: 

In the Jupyter Notebook, there is a speciall "Attachments" cell toolbar which you can use to see all attachments of a cell and delete them, if needed.

3.7 HTML Elements (HTML only)

It is allowed to use plain HTML elements within Markdown cells. Those elements are passed through to the HTML output and are ignored for the LaTeX output. Below are a few examples.

HTML5 audio[85] elements can be created like this:

```
<audio src="https://example.org/audio.ogg" controls>alternative text</audio>
```

Example:

The HTML audio element is not supported!

HTML5 video[86] elements can be created like this:

```
<video src="https://example.org/video.ogv" controls>alternative text</video>
```

Example:

The HTML video element is not supported!

[85] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio
[86] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video

The alternative text is shown in browsers that don't support those elements. The same text is also shown in Sphinx's LaTeX output.

Note: You can also use local files for the <audio> and <video> elements, but you have to create a link to the source file somewhere, because only then are the local files copied to the HTML output directory! You should do that anyway to make the audio/video file accessible to browsers that don't support the <audio> and <video> elements.

3.8 Info/Warning Boxes

> **Warning:**
>
> This is an *experimental feature*! Its usage will probably change in the future or it might be removed completely!

Until there is an info/warning extension for Markdown/CommonMark (see this issue[87]), such boxes can be created by using HTML elements like this:

```
<div class="alert alert-info">

**Note:** This is a note!

</div>
```

For this to work reliably, you should obey the following guidelines:

- The class attribute has to be either "alert alert-info" or "alert alert-warning", other values will not be converted correctly.
- No further attributes are allowed.
- For compatibility with CommonMark, you should add an empty line between the start tag and the beginning of the content.

Note:

The text can contain further Markdown formatting. It is even possible to have nested boxes:

> ... but please don't *overuse* this!

3.9 Links to Other Notebooks

Relative links to local notebooks can be used: *a link to a notebook in a subdirectory* (page 35), a link to an orphan notebook (latter won't work in LaTeX output, because orphan pages are not included there).

This is how a link is created in Markdown:

```
[a link to a notebook in a subdirectory](subdir/a-notebook-in-a-subdir.ipynb)
```

[87] https://github.com/jupyter/notebook/issues/1292

Markdown also supports *reference-style* links: *a reference-style link* (page 35), *another version of the same link* (page 35).

These can be created with this syntax:

```
[a reference-style link][mylink]

[mylink]: subdir/a-notebook-in-a-subdir.ipynb
```

Links to sub-sections are also possible, e.g. *this subsection* (page 35).

This link was created with:

```
[this subsection](subdir/a-notebook-in-a-subdir.ipynb#A-Sub-Section)
```

You just have to remember to replace spaces with hyphens!

BTW, links to sections of the current notebook work, too, e.g. *beginning of this section* (page 16).

This can be done, as expected, like this:

```
[beginning of this section](#Links-to-Other-Notebooks)
```

3.10 Links to *.rst Files (and Other Sphinx Source Files)

Links to files whose extension is in the configuration value source_suffix[88], will be converted to links to the generated HTML/LaTeX pages. Example: *A reStructuredText file* (page 37).

This was created with:

```
[A reStructuredText file](a-normal-rst-file.rst)
```

Links to sub-sections are also possible. Example: *Sphinx Directives* (page 38).

This was created with:

```
[Sphinx Directives](a-normal-rst-file.rst#sphinx-directives-for-info-warning-boxes)
```

Note:

Sphinx section anchors are different from Jupyter section anchors! To create a link to a subsection in an .rst file (or another non-notebook source file), you not only have to replace spaces with hyphens, but also slashes and some other characters. In case of doubt, just check the target HTML page generated by Sphinx.

3.11 Links to Local Files

Links to local files (other than Jupyter notebooks and other Sphinx source files) are also possible, e.g. requirements.txt.

This was simply created with:

```
[requirements.txt](requirements.txt)
```

The linked files are automatically copied to the HTML output directory. For LaTeX output, links are created, but the files are not copied to the target directory.

[88] https://www.sphinx-doc.org/en/master/config.html#confval-source_suffix

3.12 Links to Domain Objects

Links to Sphinx domain objects[89] (such as a Python class or JavaScript function) are also possible. For example: *example_python_function()* (page 39).

This was created with:

```
[example_python_function()](a-normal-rst-file.rst#example_python_function)
```

This is especially useful for use with the Sphinx autodoc[90] extension!

doc/markdown-cells.ipynb ends here.

The following section was generated from doc/code-cells.ipynb

4 Code Cells

4.1 Code, Output, Streams

An empty code cell:

[ ]:

Two empty lines:

[ ]:

Leading/trailing empty lines:

[1]:
```
# 2 empty lines before, 1 after
```

A simple output:

[2]:
```
6 * 7
```

[2]:
```
42
```

The standard output stream:

[3]:
```
print('Hello, world!')
```

```
Hello, world!
```

Normal output + standard output

[4]:
```
print('Hello, world!')
6 * 7
```

```
Hello, world!
```

[4]:
```
42
```

The standard error stream is highlighted and displayed just below the code cell. The standard output stream comes afterwards (with no special highlighting). Finally, the "normal" output is displayed.

[89] https://www.sphinx-doc.org/en/master/usage/restructuredtext/domains.html
[90] https://www.sphinx-doc.org/en/master/ext/autodoc.html

[5]:
```
import sys

print("I'll appear on the standard error stream", file=sys.stderr)
print("I'll appear on the standard output stream")
"I'm the 'normal' output"
```

```
I'll appear on the standard output stream
```

```
I'll appear on the standard error stream
```

[5]:
```
"I'm the 'normal' output"
```

Note:

Using the IPython kernel, the order is actually mixed up, see https://github.com/ipython/ipykernel/issues/280.

4.2 Cell Magics

IPython can handle code in other languages by means of cell magics[91]:

[6]:
```
%%bash
for i in 1 2 3
do
    echo $i
done
```

```
1
2
3
```

4.3 Special Display Formats

See IPython example notebook[92].

4.3.1 Local Image Files

[7]:
```
from IPython.display import Image
i = Image(filename='images/notebook_icon.png')
i
```

[7]:

[8]:
```
display(i)
```

[91] https://ipython.readthedocs.io/en/stable/interactive/magics.html#cell-magics

[92] https://nbviewer.jupyter.org/github/ipython/ipython/blob/master/examples/IPython%20Kernel/Rich%20Output.ipynb

See also *SVG support for LaTeX* (page 14).

[9]:
```
from IPython.display import SVG
SVG(filename='images/python_logo.svg')
```

[9]:


4.3.2 Image URLs

[10]:
```
Image(url='https://www.python.org/static/img/python-logo-large.png')
```

[10]:
```
<IPython.core.display.Image object>
```

[11]:
```
Image(url='https://www.python.org/static/img/python-logo-large.png', embed=True)
```

[11]:
```
<IPython.core.display.Image object>
```

[12]:
```
Image(url='https://jupyter.org/assets/nav_logo.svg')
```

[12]:
```
<IPython.core.display.Image object>
```

4.3.3 Math

[13]:
```
from IPython.display import Math
eq = Math(r'\int\limits_{-\infty}^\infty f(x) \delta(x - x_0) dx = f(x_0)')
eq
```

[13]:
$$\int\limits_{-\infty}^{\infty} f(x)\delta(x - x_0)dx = f(x_0)$$

[14]:
```
display(eq)
```

$$\int\limits_{-\infty}^{\infty} f(x)\delta(x - x_0)dx = f(x_0)$$

[15]:
```
from IPython.display import Latex
Latex(r'This is a \LaTeX{} equation: $a^2 + b^2 = c^2$')
```

[15]: This is a LaTeX equation: $a^2 + b^2 = c^2$

[16]:
```
%%latex
\begin{equation}
\int\limits_{-\infty}^\infty f(x) \delta(x - x_0) dx = f(x_0)
\end{equation}
```

$$\int_{-\infty}^{\infty} f(x)\delta(x - x_0)dx = f(x_0) \tag{3}$$

4.3.4 Plots

The output formats for Matplotlib plots can be customized. You'll need separate settings for the Jupyter Notebook application and for nbsphinx.

If you want to use SVG images for Matplotlib plots, add this line to your IPython configuration file:

```
c.InlineBackend.figure_formats = {'svg'}
```

If you want SVG images, but also want nice plots when exporting to LaTeX/PDF, you can select:

```
c.InlineBackend.figure_formats = {'svg', 'pdf'}
```

If you want to use the default PNG plots or HiDPI plots using 'png2x' (a.k.a. 'retina'), make sure to set this:

```
c.InlineBackend.rc = {'figure.dpi': 96}
```

This is needed because the default 'figure.dpi' value of 72 is only valid for the Qt Console[93].

If you are planning to store your SVG plots as part of your notebooks, you should also have a look at the 'svg.hashsalt' setting.

For more details on these and other settings, have a look at Default Values for Matplotlib's "inline" Backend[94].

The configuration file ipython_kernel_config.py can be either in the directory where your notebook is located (see the ipython_kernel_config.py in this directory), or in your profile directory (typically ~/.ipython/profile_default/ipython_kernel_config.py). To find out your IPython profile directory, use this command:

```
python3 -m IPython profile locate
```

A local ipython_kernel_config.py in the notebook directory also works on https://mybinder.org/. Alternatively, you can create a file with those settings in a file named .ipython/profile_default/ipython_kernel_config.py in your repository.

To get SVG and PDF plots for nbsphinx, use something like this in your conf.py file:

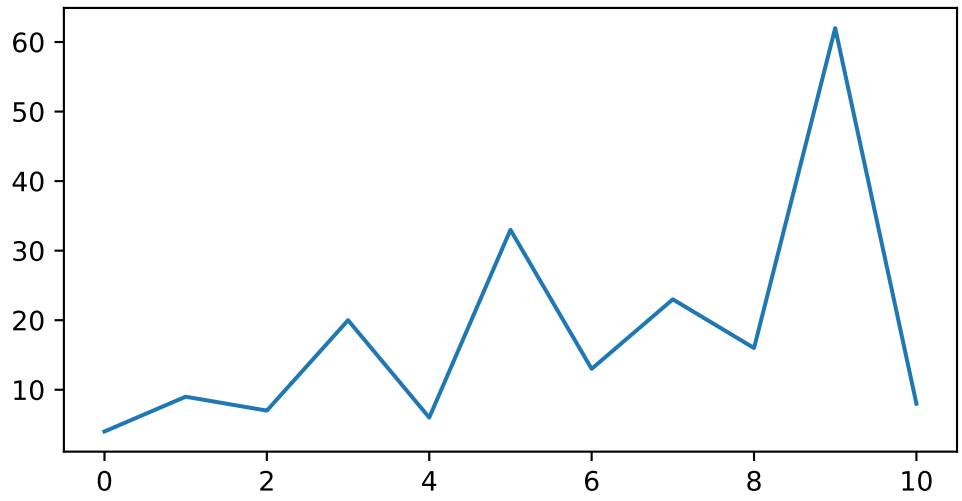
```
nbsphinx_execute_arguments = [
    "--InlineBackend.figure_formats={'svg', 'pdf'}",
    "--InlineBackend.rc={'figure.dpi': 96}",
]
```

[93] https://qtconsole.readthedocs.io/
[94] https://nbviewer.jupyter.org/github/mgeier/python-audio/blob/master/plotting/matplotlib-inline-defaults.ipynb

In the following example, nbsphinx should use an SVG image in the HTML output and a PDF image for LaTeX/PDF output.

[17]:
```
import matplotlib.pyplot as plt
```

[18]:
```
fig, ax = plt.subplots(figsize=[6, 3])
ax.plot([4, 9, 7, 20, 6, 33, 13, 23, 16, 62, 8]);
```

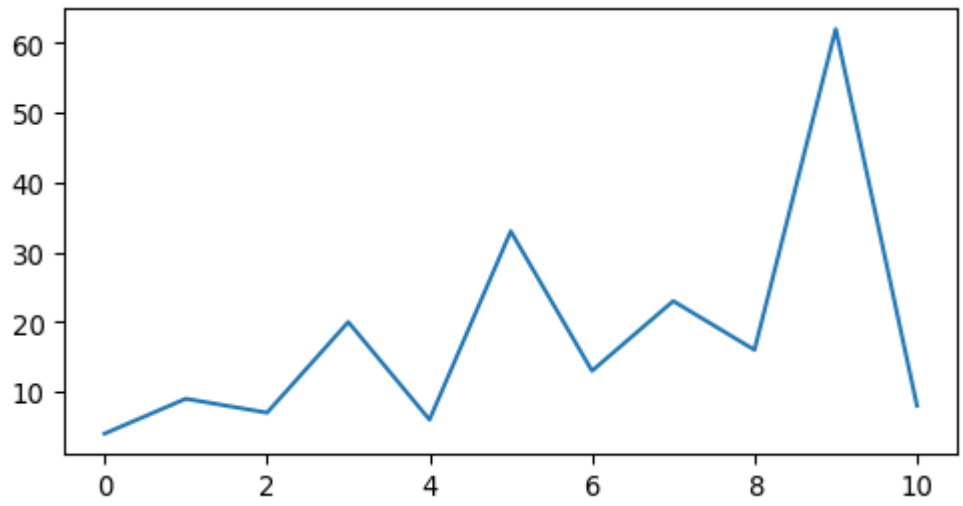


Alternatively, the figure format(s) can also be chosen directly in the notebook (which overrides the setting in nbsphinx_execute_arguments and in the IPython configuration):

[19]:
```
%config InlineBackend.figure_formats = ['png']
```

[20]:
```
fig
```

[20]:

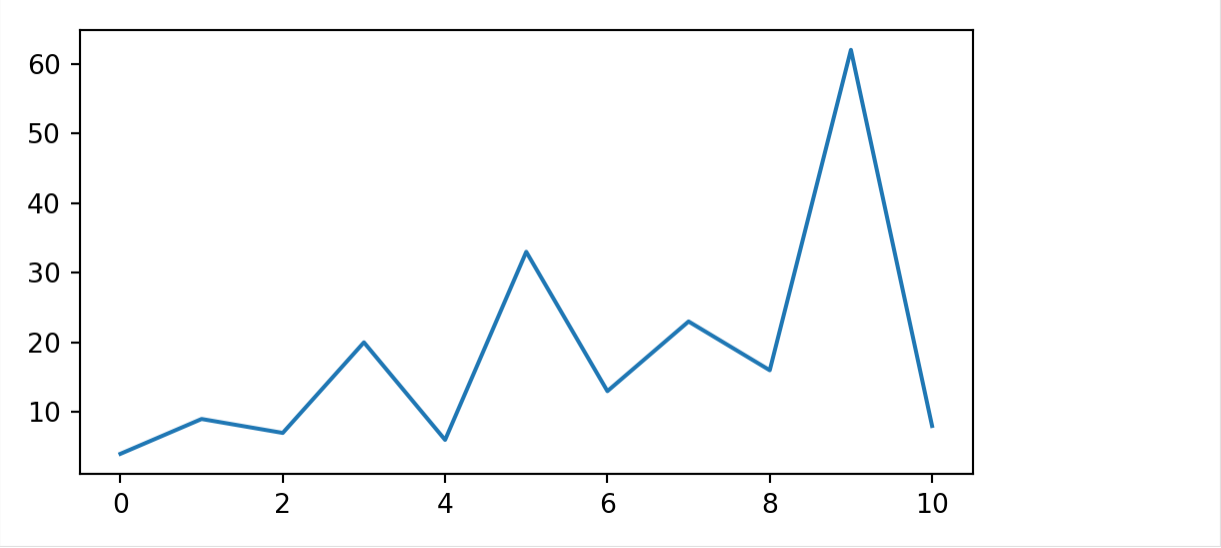


If you want to use PNG images, but with HiDPI resolution, use the special 'png2x' (a.k.a. 'retina') format (which also looks nice in the LaTeX output):

[21]:
```
%config InlineBackend.figure_formats = ['png2x']
```

[22]:
```
fig
```

[22]:



4.3.5 Pandas Dataframes

Pandas dataframes[95] should be displayed as nicely formatted HTML tables (if you are using HTML output).

[23]:
```
import numpy as np
import pandas as pd
```

[24]:
```
df = pd.DataFrame(np.random.randint(0, 100, size=[5, 4]),
                  columns=['a', 'b', 'c', 'd'])
df
```

[24]:

	a	b	c	d
0	56	45	11	84
1	58	63	0	63
2	24	7	49	62
3	8	56	42	73
4	37	33	28	83

For LaTeX output, however, the plain text output is used by default.

To get nice LaTeX tables, a few settings have to be changed:

[25]:
```
pd.set_option('display.latex.repr', True)
```

This is not enabled by default because of Pandas issue #12182[96].

The generated LaTeX tables utilize the booktabs package, so you have to make sure that package is loaded in the preamble[97] with:

```
\usepackage{booktabs}
```

In order to allow page breaks within tables, you should use:

[26]:
```
pd.set_option('display.latex.longtable', True)
```

[95] https://pandas.pydata.org/pandas-docs/stable/getting_started/dsintro.html#dataframe
[96] https://github.com/pandas-dev/pandas/issues/12182
[97] https://www.sphinx-doc.org/en/master/latex.html

The longtable package is already used by Sphinx, so you don't have to manually load it in the preamble.

Finally, if you want to use LaTeX math expressions in your dataframe, you'll have to disable escaping:

[27]:
```
pd.set_option('display.latex.escape', False)
```

The above settings should have no influence on the HTML output, but the LaTeX output should now look nicer:

[28]:
```
df = pd.DataFrame(np.random.randint(0, 100, size=[10, 4]),
                  columns=[r'$\alpha$', r'$\beta$', r'$\gamma$', r'$\delta$'])
df
```

[28]:

	α	β	γ	δ
0	18	88	34	70
1	5	81	32	85
2	41	71	24	22
3	3	10	17	36
4	16	41	25	95
5	8	36	65	86
6	32	91	94	59
7	17	50	54	40
8	53	47	72	21
9	99	57	11	67

4.3.6 YouTube Videos

[29]:
```
from IPython.display import YouTubeVideo
YouTubeVideo('WAikxUGbomY')
```

[29]:
```
<IPython.lib.display.YouTubeVideo at 0x7fe657331690>
```

4.3.7 Arbitrary JavaScript Output (HTML only)

[30]:
```
%%javascript

var text = document.createTextNode("Hello, I was generated with JavaScript!");
// Content appended to "element" will be visible in the output area:
element.appendChild(text);
```

```
<IPython.core.display.Javascript object>
```

4.3.8 Unsupported Output Types

If a code cell produces data with an unsupported MIME type, the Jupyter Notebook doesn't generate any output. nbsphinx, however, shows a warning message.

[31]:
```
display({
    'text/x-python': 'print("Hello, world!")',
    'text/x-haskell': 'main = putStrLn "Hello, world!"',
}, raw=True)
```

(continued from previous page)

Data type cannot be displayed: text/x-python, text/x-haskell

4.4 ANSI Colors

The standard output and standard error streams may contain ANSI escape sequences[98] to change the text and background colors.

[32]:
```
print('BEWARE: \x1b[1;33;41mugly colors\x1b[m!', file=sys.stderr)
print('AB\x1b[43mCD\x1b[35mEF\x1b[1mGH\x1b[4mIJ\x1b[7m'
      'KL\x1b[49mMN\x1b[39mOP\x1b[22mQR\x1b[24mST\x1b[27mUV')
```

```
ABCDEFGHIJKLMNOPQRSTUV
```

```
BEWARE: ugly colors!
```

The following code showing the 8 basic ANSI colors is based on http://tldp.org/HOWTO/Bash-Prompt-HOWTO/x329.html. Each of the 8 colors has an "intense" variation, which is used for bold text.

[33]:
```
text = ' XYZ '
formatstring = '\x1b[{}m' + text + '\x1b[m'

print('    ' + '    ' * len(text) +
      ''.join('{:^{}}'.format(bg, len(text)) for bg in range(40, 48)))
for fg in range(30, 38):
    for bold in False, True:
        fg_code = ('1;' if bold else '') + str(fg)
        print(' {:>4} '.format(fg_code) + formatstring.format(fg_code) +
              ''.join(formatstring.format(fg_code + ';' + str(bg))
                      for bg in range(40, 48)))
```

```
        40   41   42   43   44   45   46   47
  30    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
1;30    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
  31    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
1;31    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
  32    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
1;32    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
  33    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
1;33    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
  34    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
1;34    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
  35    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
1;35    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
  36    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
1;36    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
  37    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
1;37    XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
```

ANSI also supports a set of 256 indexed colors. The following code showing all of them is based on http://bitmote.com/index.php?post/2012/11/19/Using-ANSI-Color-Codes-to-Colorize-Your-Bash-Prompt-on-Linux[99].

[98] https://en.wikipedia.org/wiki/ANSI_escape_code
[99] https://web.archive.org/web/20190109005413/http://bitmote.com/index.php?post/2012/11/19/Using-ANSI-Color-Codes-to-Colorize-Your-Bash-Prompt-on-Linux

[34]:
```
formatstring = '\x1b[38;5;{0};48;5;{0}mX\x1b[1mX\x1b[m'

print('   ' + ' '.join('{:2}'.format(i) for i in range(36)))
print('  0 ' + ''.join(formatstring.format(i) for i in range(16)))
for i in range(7):
    i = i * 36 + 16
    print('{:3} '.format(i) + ''.join(formatstring.format(i + j)
                                      for j in range(36) if i + j < 256))
```

```
   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
  0 X X X X X X X X X X
 16
 52
 88
124
160
196
232
```

You can even use 24-bit RGB colors:

[35]:
```
start = 255, 0, 0
end = 0, 0, 255
length = 79
out = []

for i in range(length):
    rgb = [start[c] + int(i * (end[c] - start[c]) / length) for c in range(3)]
    out.append('\x1b['
               '38;2;{rgb[2]};{rgb[1]};{rgb[0]};'
               '48;2;{rgb[0]};{rgb[1]};{rgb[2]}mX\x1b[m'.format(rgb=rgb))
print(''.join(out))
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

doc/code-cells.ipynb ends here.

The following section was generated from doc/raw-cells.ipynb

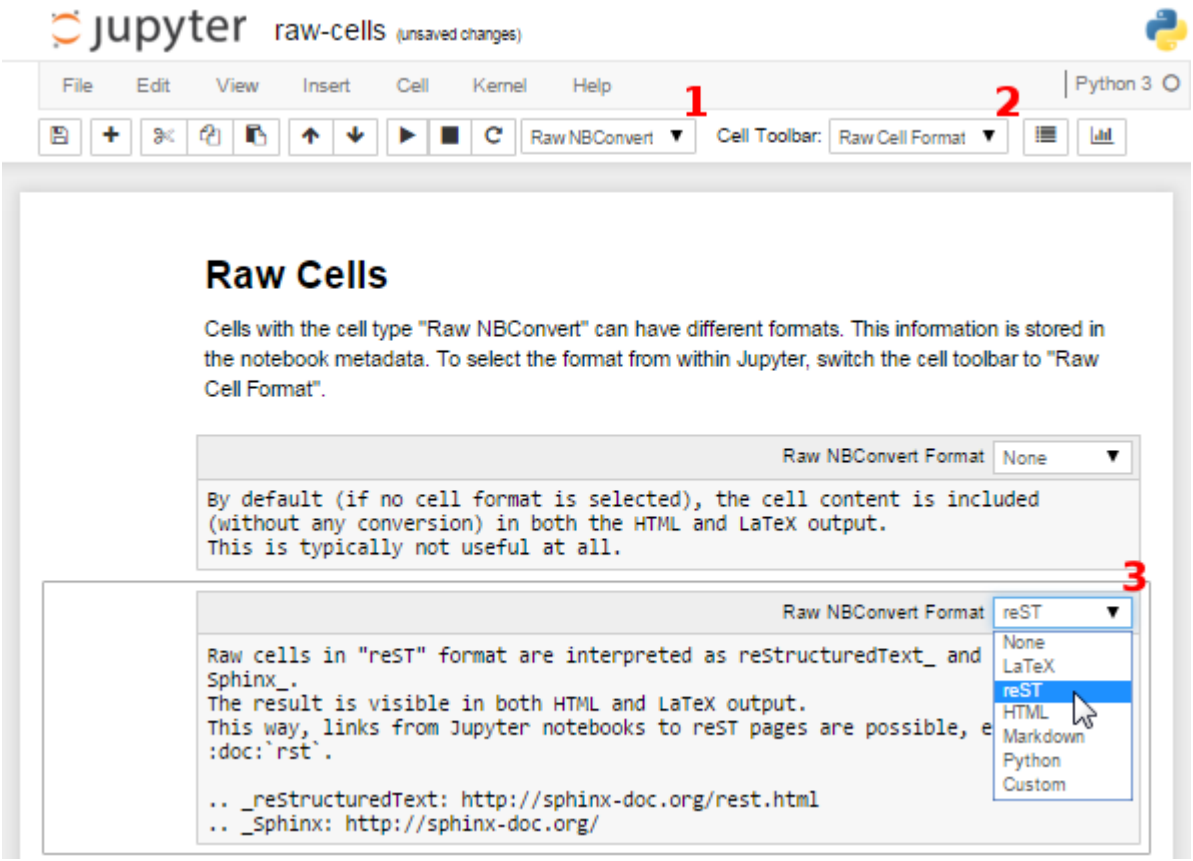
5 Raw Cells

The “Raw NBConvert” cell type can be used to render different code formats into HTML or LaTeX by Sphinx. This information is stored in the notebook metadata and converted appropriately.

5.1 Usage

To select a desired format from within Jupyter, select the cell containing your special code and choose options from the following dropdown menus:

1. Select “Raw NBConvert”
2. Switch the Cell Toolbar to “Raw Cell Format”
3. Chose the appropriate “Raw NBConvert Format” within the cell



5.2 Available Raw Cell Formats

The following examples show how different Jupyter cell formats are rendered by Sphinx.

5.2.1 None

By default (if no cell format is selected), the cell content is included (without any conversion) in both the HTML and LaTeX output. This is typically not useful at all.

"I'm a raw cell with no format."

5.2.2 reST

Raw cells in "reST" format are interpreted as reStructuredText and parsed by Sphinx. The result is visible in both HTML and LaTeX output.

"**I'm** a *raw cell* in reST[100] format."

5.2.3 Markdown

Raw cells in "Markdown" format are interpreted as Markdown, and the result is included in both HTML and LaTeX output. Since the Jupyter Notebook also supports normal Markdown cells, this might not be useful *at all*.

"**I'm** a *raw cell* in Markdown[101] format."

[100] https://www.sphinx-doc.org/rest.html
[101] https://daringfireball.net/projects/markdown/

5.2.4 HTML

Raw cells in "HTML" format are only visible in HTML output. This option might not be very useful, since raw HTML code is also allowed within normal Markdown cells.

5.2.5 LaTeX

Raw cells in "LaTeX" format are only visible in LaTeX output.

I'm a *raw cell* in LaTeX format.

5.2.6 Python

Raw cells in "Python" format are not visible at all (nor executed in any way).

doc/raw-cells.ipynb ends here.

The following section was generated from doc/hidden-cells.ipynb

6 Hidden Cells

You can remove cells from the HTML/LaTeX output by adding this to the cell metadata:

```
"nbsphinx": "hidden"
```

Hidden cells are still executed but removed afterwards.

For example, the following hidden cell defines the variable answer.

This is the cell after the hidden cell. Although the previous cell is not visible, its result is still available:

[2]:
```
answer
```

[2]:
```
42
```

Don't overuse this, because it may make it harder to follow what's going on in your notebook.

Also Markdown cells can be hidden. The following cell is hidden.

This is the cell after the hidden cell.

doc/hidden-cells.ipynb ends here.

The following section was generated from doc/executing-notebooks.ipynb

7 Controlling Notebook Execution

Notebooks with no outputs are automatically executed during the Sphinx build process. If, however, there is at least one output cell present, the notebook is not evaluated and included as is.

The following notebooks show how this default behavior can be used and customized.

The following section was generated from doc/pre-executed.ipynb

7.1 Pre-Executing Notebooks

Automatically executing notebooks during the Sphinx build process is an important feature of nbsphinx. However, there are a few use cases where pre-executing a notebook and storing the outputs might be preferable. Storing any output will, by default, stop nbsphinx from executing the notebook.

7.1.1 Long-Running Cells

If you are doing some very time-consuming computations, it might not be feasible to re-execute the notebook every time you build your Sphinx documentation.

So just do it once – when you happen to have the time – and then just keep the output.

[1]:
```
import time
```

[2]:
```
%time time.sleep(60 * 60)
6 * 7
```

```
CPU times: user 160 ms, sys: 56 ms, total: 216 ms
Wall time: 1h 1s
```

[2]:
```
42
```

If you *do* want to execute your notebooks, but some cells run for a long time, you can change the timeout, see *Cell Execution Timeout* (page 32).

7.1.2 Rare Libraries

You might have created results with a library that's hard to install and therefore you have only managed to install it on one very old computer in the basement, so you probably cannot run this whenever you build your Sphinx docs.

[3]:
```
from a_very_rare_library import calculate_the_answer
```

[4]:
```
calculate_the_answer()
```

[4]:
```
42
```

7.1.3 Exceptions

If an exception is raised during the Sphinx build process, it is stopped (the build process, not the exception!). If you want to show to your audience how an exception looks like, you have two choices:

1. Allow errors – either generally or on a per-notebook or per-cell basis – see *Ignoring Errors* (page 30) (*per cell* (page 31)).
2. Execute the notebook beforehand and save the results, like it's done in this example notebook:

[5]:
```
1 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-5-b710d87c980c> in <module>()
--> 1 1 / 0

ZeroDivisionError: division by zero
```

doc/pre-executed.ipynb ends here.

The following section was generated from doc/never-execute.ipynb

7.2 Explicitly Dis-/Enabling Notebook Execution

If you want to include a notebook without outputs and yet don't want nbsphinx to execute it for you, you can explicitly disable this feature.

You can do this globally by setting the following option in conf.py:

```
nbsphinx_execute = 'never'
```

Or on a per-notebook basis by adding this to the notebook's JSON metadata:

```
"nbsphinx": {
  "execute": "never"
},
```

There are three possible settings, "always", "auto" and "never". By default (= "auto"), notebooks with no outputs are executed and notebooks with at least one output are not. As always, per-notebook settings take precedence over the settings in conf.py.

This very notebook has its metadata set to "never", therefore the following cell is not executed:

[ ]:
```
6 * 7
```

doc/never-execute.ipynb ends here.

The following section was generated from doc/allow-errors.ipynb

7.3 Ignoring Errors

Normally, if an exception is raised while executing a notebook, the Sphinx build process is stopped immediately.

If a notebook contains errors on purpose (or if you are too lazy to fix them right now), you have four options:

1. Manually execute the notebook in question and save the results, see *the pre-executed example notebook* (page 28).
2. Allow errors in all notebooks by setting this option in conf.py:

   ```
   nbsphinx_allow_errors = True
   ```

3. Allow errors on a per-notebook basis by adding this to the notebook's JSON metadata:

   ```
   "nbsphinx": {
     "allow_errors": true
   },
   ```

4. Allow errors on a per-cell basis using the raises-exception tag, see *Ignoring Errors on a Cell-by-Cell Basis* (page 31).

This very notebook is an example for the third option. The results of the following code cells are not stored within the notebook, therefore it is executed during the Sphinx build process. Since the above-mentioned allow_errors flag is set in this notebook's metadata, all cells are executed although most of them cause an exception.

[1]:
```
nonsense
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-7dd4c0df649c> in <module>()
---> 1 nonsense

NameError: name 'nonsense' is not defined
```

[2]:
```
42 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-52cebea8b64f> in <module>()
---> 1 42 / 0

ZeroDivisionError: division by zero
```

[3]:
```
print 'Hello, world!'
```

```
  File "<ipython-input-3-653b30cd70a8>", line 1
    print 'Hello, world!'
                        ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print('Hello, world!')?
```

[4]:
```
6 ~ 7
```

```
  File "<ipython-input-4-8300b2622db3>", line 1
    6 ~ 7
      ^
SyntaxError: invalid syntax
```

[5]:
```
6 * 7
```

[5]:
```
42
```

doc/allow-errors.ipynb ends here.

The following section was generated from doc/allow-errors-per-cell.ipynb

7.4 Ignoring Errors on a Per-Cell Basis

Instead of ignoring errors for all notebooks or for some selected notebooks (see *the previous notebook* (page 30)), you can be more fine-grained and just allow errors on certain code cells by tagging them with the raises-exception tag.

[1]:
```
'no problem'
```

[1]:
```
'no problem'
```

The following code cell has the raises-exception tag.

[2]:
```
problem
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-526ab3a89ffc> in <module>()
---> 1 problem

NameError: name 'problem' is not defined
```

The following code cell is executed even though the previous cell raised an exception.

[3]:
```
'no problem'
```

[3]:
```
'no problem'
```

Note:

The behavior of the raises-exception tag doesn't match its name. While it does *allow* exceptions, it does not check if an exception is actually raised!

This will hopefully be fixed at some point, see https://github.com/jupyter/nbconvert/issues/730.

doc/allow-errors-per-cell.ipynb ends here.

The following section was generated from doc/timeout.ipynb

7.5 Cell Execution Timeout

By default, nbconvert (which is used to execute the notebooks during the Sphinx build process) will give a cell 30 seconds to execute before it times out.

If you would like to change the amount of time given for a cell, you can change the timeout length for all notebooks by setting the following option in conf.py:

```
nbsphinx_timeout = 60
```

Or change the timeout length on a per-notebook basis by adding this to the notebook's JSON metadata:

```
"nbsphinx": {
  "timeout": 60
},
```

The timeout is given in seconds, use -1 to disable the timeout.

Alternatively, you can manually execute the notebook in question and save the results, see *the pre-executed example notebook* (page 28).

doc/timeout.ipynb ends here.

doc/executing-notebooks.ipynb ends here.

The following section was generated from doc/prolog-and-epilog.ipynb

8 Prolog and Epilog

When including notebooks in your Sphinx documentation, you can choose to add some generic content before and after each notebook. This can be done with the configuration values nbsphinx_prolog and nbsphinx_epilog in the file conf.py.

The prolog and epilog strings can hold arbitrary reST[102] markup. Particularly, the only[103] and raw[104] directives can be used to have different content for HTML and LaTeX output.

Those strings are also processed by the Jinja2[105] templating engine. This means you can run Python-like code within those strings. You have access to the current Sphinx build environment[106] via the variable env. Most notably, you can get the file name of the current notebook with

```
{{ env.doc2path(env.docname, base=None) }}
```

Have a look at the Jinja2 template documentation[107] for more information.

[102] https://www.sphinx-doc.org/rest.html
[103] https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-only
[104] https://docutils.readthedocs.io/en/sphinx-docs/ref/rst/directives.html#raw-data-pass-through
[105] http://jinja.pocoo.org/
[106] https://www.sphinx-doc.org/en/master/extdev/envapi.html
[107] http://jinja.pocoo.org/docs/latest/templates/

Warning:

If you use invalid syntax, you might get an error like this:

```
jinja2.exceptions.TemplateSyntaxError: expected token ':', got '}'
```

This is especially prone to happen when using raw LaTeX, with its abundance of braces. To avoid clashing braces you can try to insert additional spaces or LaTeX macros that don't have a visible effect, like e.g. `\strut{}`. For example, you can avoid three consecutive opening braces with something like that:

```
\texttt{\strut{}{{ env.doc2path(env.docname, base=None) }}}
```

NB: The three consecutive closing braces in this example are not problematic.

An alternative work-around would be to surround LaTeX braces with Jinja braces like this:

```
{{ '{' }}
```

The string within will not be touched by Jinja.

Another special Jinja syntax is {%, which is also often used in fancy TeX/LaTeX code. A work-around for this situation would be to use

```
{{ '{%' }}
```

8.1 Examples

You can include a simple static string, using reST[108] markup if you like:

```
nbsphinx_epilog = """
----

Generated by nbsphinx_ from a Jupyter_ notebook.

.. _nbsphinx: https://nbsphinx.readthedocs.io/
.. _Jupyter: https://jupyter.org/
"""
```

Using some additional Jinja2 markup and the information from the `env` variable, you can create URLs that point to the current notebook file, but located on some other server:

```
nbsphinx_prolog = """
Go there: https://example.org/notebooks/{{ env.doc2path(env.docname, base=None) }}

----
"""
```

You can also use separate content for HTML and LaTeX output, e.g.:

```
nbsphinx_prolog = r"""
{% set docname = env.doc2path(env.docname, base=None) %}

.. only:: html

    Go there: https://example.org/notebooks/{{ docname }}

.. raw:: latex
```

(continues on next page)

[108] https://www.sphinx-doc.org/rest.html

(continued from previous page)

```
    \nbsphinxstartnotebook{The following section was created from
    \texttt{\strut{}{{ docname }}:}
"""

nbsphinx_epilog = r"""
.. raw:: latex

    \nbsphinxstopnotebook{\hfill End of notebook.}
"""
```

Note the use of the \nbsphinxstartnotebook and \nbsphinxstopnotebook commands. Those make sure there is not too much space between the "prolog" and the beginning of the notebook and, respectively, between the end of the notebook and the "epilog". They also avoid page breaks, in order for the "prolog"/"epilog" not to end up on the page before/after the notebook.

For a more involved example for different HTML and LaTeX versions, see the file conf.py of the nbsphinx documentation.

doc/prolog-and-epilog.ipynb ends here.

The following section was generated from doc/custom-formats.ipynb

9 Custom Notebook Formats

By default, Jupyter notebooks are stored in files with the suffix .ipynb, which use the JSON format for storage.

However, there are libraries available which allow storing notebooks in different formats, using different file suffixes.

To use a custom format in nbsphinx, you can specify the nbsphinx_custom_formats option in your conf.py file. You have to provide the file extension and a conversion function that takes the contents of a file (as a string) and returns a Jupyter notebook object.

```
nbsphinx_custom_formats = {
    '.mysuffix': 'mylibrary.converter_function',
}
```

The converter function can be given as a string or as a function object.

One example for such library is jupytext[109], which allows storing the contents of Jupyter notebooks in Markdown and R-Markdown, as well as plain Julia, Python and R files.

Since its conversion function takes more than a single string argument, just using the function name 'jupytext.reads' will not work. We have to create a function object, and one way to do that is using a lambda function like this:

```
import jupytext

nbsphinx_custom_formats = {
    '.Rmd': lambda s: jupytext.reads(s, '.Rmd'),
}
```

You can of course use multiple formats by specifying multiple conversion functions.

doc/custom-formats.ipynb ends here.

[109] https://github.com/mwouts/jupytext

The following section was generated from doc/subdir/a-notebook-in-a-subdir.ipynb

10 Notebooks in Sub-Directories

You can organize your notebooks in subdirectories and nbsphinx will take care that relative links to other notebooks, images and other files still work.

Let's see if links to local images work:

[1]:
```
from IPython.display import Image
Image(filename='../images/notebook_icon.png')
```

[1]:

Warning:

There may be problems with images in output cells if your source directory contains symbolic links, see issue #49[110].

A link to a notebook in the same sub-directory: *link* (page 35).

A link to a notebook in the parent directory: *link* (page 11).

A link to a local file: link.

A random equation:

$$F_n = F_{n-1} + F_{n-2} \tag{08.15}$$

10.1 A Sub-Section

This is just for testing inter-notebook links, see *this section* (page 16).

doc/subdir/a-notebook-in-a-subdir.ipynb ends here.

The following section was generated from doc/subdir/toctree.ipynb

11 Using toctree In A Notebook

In Sphinx-based documentation, there is typically a file called index.rst which contains one or more toctree[111] directives. Those can be used to pull in further source files (which themselves can contain toctree directives).

[110] https://github.com/spatialaudio/nbsphinx/issues/49
[111] https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-toctree

With nbsphinx it is possible to get a similar effect within a Jupyter notebook using the "nbsphinx-toctree" cell metadata. Markdown cells with "nbsphinx-toctree" metadata are not converted like "normal" Markdown cells. Instead, they are only scanned for links to other notebooks (or *.rst files and other Sphinx source files) and those links are added to a toctree directive. External links can also be used, but they will not be visible in the LaTeX output.

If there is a section title in the cell, it is used as toctree caption (but it also works without a title).

Note:

All other content of such a cell is *ignored*!

Use ...

```
"nbsphinx-toctree": {}
```

... for the default settings, ...

```
"nbsphinx-toctree": {
  "maxdepth": 2
}
```

... for setting the :maxdepth: option, or...

```
"nbsphinx-toctree": {
  "hidden": true
}
```

... for setting the :hidden: option.

Of course, multiple options can be used at the same time, e.g.

```
"nbsphinx-toctree": {
  "maxdepth": 3,
  "numbered": true
}
```

For more options, have a look a the Sphinx documentation[112]. All options can be used – except :glob:, which can only be used in *rst files* (page 37) and in *raw reST cells* (page 27).

Note that in the HTML output, a toctree cell generates an in-line table of contents (containing links) at its position in the notebook, whereas in the LaTeX output, a new (sub-)section with the actual content is inserted at its position. All content below the toctree cell will appear after the table of contents/inserted section, respectively. If you want to use the LaTeX output, it is recommended that you don't add further cells below a toctree cell, otherwise their content may appear at unexpected places. Multiple toctree cells in a row should be fine, though.

The following cell is tagged with "nbsphinx-toctree" metadata and contains a link to the notebook *yet-another.ipynb* (page 37) and an external link (which will only be visible in the HTML output). It also contains a section title which will be used as toctree caption (which also will only be visible in the HTML output).

[112] https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#directive-toctree

The following section was generated from doc/yet-another.ipynb

11.1 Yet Another Notebook

This notebook is only here to show how (sub-)toctrees can be created with Markdown cell metadata. See *there* (page 35).

doc/yet-another.ipynb ends here.

doc/subdir/toctree.ipynb ends here.

12 Normal reStructuredText Files

This is a normal RST file.

Note: Those still work!

12.1 Links to Notebooks (and Other Sphinx Source Files)

Links to Sphinx source files can be created like normal Sphinx hyperlinks[113], just using a relative path to the local file: *link* (page 35).

```
using a relative path to the local file: link_.

.. _link: subdir/a-notebook-in-a-subdir.ipynb
```

If the link text has a space (or some other strange character) in it, you have to surround it with backticks: *a notebook link* (page 35).

```
surround it with backticks: `a notebook link`_.

.. _a notebook link: subdir/a-notebook-in-a-subdir.ipynb
```

You can also use an anonymous hyperlink target[114], like this: *link* (page 35). If you have multiple of those, their order matters!

```
like this: link__.

__ subdir/a-notebook-in-a-subdir.ipynb
```

Finally, you can use Embedded URIs[115], like this *link* (page 35).

```
like this `link <subdir/a-notebook-in-a-subdir.ipynb>`_.
```

Note: These links should also work on Github and in other rendered reStructuredText pages.

Links to subsections are also possible by adding a hash sign (#) and the section title to any of the above-mentioned link variants. You have to replace spaces in the section titles by hyphens. For example, see this *subsection* (page 35).

```
For example, see this subsection_.

.. _subsection: subdir/a-notebook-in-a-subdir.ipynb#A-Sub-Section
```

[113] https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html#external-links
[114] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#anonymous-hyperlinks
[115] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#embedded-uris-and-aliases

12.2 Links to Notebooks, Ye Olde Way

In addition to the way shown above, you can also create links to notebooks (and other Sphinx source files) with :ref:[116]. This has some disadvantages:

- It is arguably a bit more clunky.
- Because :ref: is a Sphinx feature, the links don't work on Github and other rendered reStructuredText pages that use plain old docutils.

It also has one important advantage:

- The link text can automatically be taken from the actual section title.

A link with automatic title looks like this: *Notebooks in Sub-Directories* (page 35).

```
:ref:`/subdir/a-notebook-in-a-subdir.ipynb`
```

But you can also provide *your own link title* (page 35).

```
:ref:`your own link title </subdir/a-notebook-in-a-subdir.ipynb>`
```

However, if you want to use your own title, you are probably better off using the method described above in *Links to Notebooks (and Other Sphinx Source Files)* (page 37).

Links to subsections are also possible, e.g. *A Sub-Section* (page 35) (the subsection title is used as link text) and *alternative text* (page 35).

These links were created with:

```
:ref:`/subdir/a-notebook-in-a-subdir.ipynb#A-Sub-Section`
:ref:`alternative text </subdir/a-notebook-in-a-subdir.ipynb#A-Sub-Section>`
```

Note:

- The paths have to be relative to the top source directory and they have to start with a slash (/).
- Spaces in the section title have to be replaced by hyphens!

12.3 Sphinx Directives for Info/Warning Boxes

> **Warning:**
>
> This is an experimental feature! Its usage may change in the future or it might disappear completely, so don't use it for now.

With a bit of luck, it will be possible (some time in the future) to create info/warning boxes in Markdown cells, see https://github.com/jupyter/notebook/issues/1292. If this ever happens, nbsphinx will provide directives for creating such boxes. For now, there are two directives available: nbinfo and nbwarning. This is how an info box looks like:

Note:

This is an info box.

It may include nested formatting, even another info/warning box:

[116] https://www.sphinx-doc.org/en/master/usage/restructuredtext/roles.html#role-ref

Warning: You should probably not use nested boxes!

12.4 Domain Objects

example_python_function(*foo*)

This is just for testing domain object links. See *this section* (page 18).

Parameters foo (str) – Example string parameter

12.5 Citations

You could use standard Sphinx citations[117], but it might be more practical to use the sphinxcontrib.bibtex[118] extension.

If you install and enable this extension, you can create citations like [PGH11]:

```
:cite:`perez2011python`
```

You can create similar citations in Jupyter notebooks with a special HTML syntax, see the section about *citations in Markdown cells* (page 13).

For those citations to work, you also need to specify a BibTeX file, as explained in the next section.

12.6 References

After installing and enabling the sphinxcontrib.bibtex[119] extension, you can create a list of references from a BibTeX file like this:

```
.. bibliography:: references.bib
```

Have a look at the documentation for all the available options.

The list of references may look something like this:

However, in the LaTeX/PDF output the list of references will not appear here, but at the end of the document. For a possible work-around, see https://github.com/mcmtroffaes/sphinxcontrib-bibtex/issues/156.

There is an alternative Sphinx extension for creating bibliographies: https://bitbucket.org/wnielson/sphinx-natbib/. However, this project seems to be abandoned (last commit in 2011).

The following section was generated from doc/links.ipynb

13 External Links

nbconvert

The official conversion tool of the Jupyter project. It can be used to convert notebooks to HTML, LaTeX and many other formats.

Its --execute flag can be used to automatically execute notebooks before conversion.

[117] https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html#citations
[118] https://sphinxcontrib-bibtex.readthedocs.io/
[119] https://sphinxcontrib-bibtex.readthedocs.io/

https://nbconvert.readthedocs.io/

https://github.com/jupyter/nbconvert

notebook_sphinxext.py

Notebooks can be included in *.rst files with a custom notebook directive. Uses runipy to execute notebooks and nbconvert to convert the result to HTML.

No LaTeX support.

https://github.com/ngoldbaum/RunNotebook

https://bitbucket.org/yt_analysis/yt-doc/src/default/extensions/notebook_sphinxext.py

https://github.com/matthew-brett/perrin-academy/blob/master/sphinxext/notebook_sphinxext.py

ipypublish

A workflow for creating and editing publication ready scientific reports and presentations, from one or more Jupyter Notebooks, without leaving the browser!

https://ipypublish.readthedocs.io/

https://github.com/chrisjsewell/ipypublish

jupyter-book

Create an online book with Jupyter Notebooks and Jekyll

https://jupyter.org/jupyter-book

https://github.com/jupyter/jupyter-book

nbinteract

Create interactive webpages from Jupyter Notebooks

https://www.nbinteract.com/

https://github.com/SamLau95/nbinteract

nb_pdf_template

An extended nbconvert template for LaTeX output.

https://github.com/t-makaro/nb_pdf_template

nb2plots

Notebook to reStructuredText converter which uses a modified version of the matplotlib plot directive.

https://github.com/matthew-brett/nb2plots

brole

A Sphinx role for IPython notebooks

https://github.com/matthew-brett/brole

Sphinx-Gallery

https://sphinx-gallery.readthedocs.io/

sphinx-nbexamples

https://sphinx-nbexamples.readthedocs.io/

https://github.com/Chilipp/sphinx-nbexamples

nbsphinx-link

https://github.com/vidartf/nbsphinx-link

Uses nbsphinx, but supports notebooks outside the Sphinx source directory.

See https://github.com/spatialaudio/nbsphinx/pull/33 for some limitations.

bookbook

Uses nbconvert to create a sequence of HTML or a concatenated LaTeX file from a sequence of notebooks.

https://github.com/takluyver/bookbook

jupyter-sphinx

Jupyter Sphinx is a Sphinx extension that executes embedded code in a Jupyter kernel, and embeds outputs of that code in the output document. It has support for rich output such as images, Latex math and even javascript widgets.

https://jupyter-sphinx.readthedocs.io/

https://github.com/jupyter/jupyter-sphinx

DocOnce

http://hplgit.github.io/doconce/doc/web/index.html

Converting Notebooks to reStructuredText

https://github.com/perrette/dimarray/blob/master/docs/scripts/nbconvert_to_rst.py

https://gist.github.com/hadim/16e29b5848672e2e497c (not available anymore)

https://sphinx-ipynb.readthedocs.io/

Converting reStructuredText to Notebooks

https://github.com/nthiery/rst-to-ipynb

https://github.com/QuantEcon/sphinxcontrib-jupyter

Converting Notebooks to HTML for Blog Posts

http://dongweiming.github.io/divingintoipynb_nikola/posts/nbconvert.html

https://github.com/getpelican/pelican-plugins/blob/master/liquid_tags/notebook.py

Further Posts and Issues

https://github.com/ipython/ipython/issues/4936

https://mail.scipy.org/pipermail/ipython-user/2013-December/013490.html (not available anymore)

doc/links.ipynb ends here.

References

[KRKP+16] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. Jupyter Notebooks—a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016. doi:10.3233/978-1-61499-649-1-87[120].

[PGH11] Fernando Pérez, Brian E. Granger, and John D. Hunter. Python: an ecosystem for scientific computing. *Computing in Science Engineering*, 13(2):13–21, 2011. doi:10.1109/MCSE.2010.119[121].

[120] https://doi.org/10.3233/978-1-61499-649-1-87
[121] https://doi.org/10.1109/MCSE.2010.119