nbsphinx

Release 0.2.13

Matthias Geier

Feb 24, 2017

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.

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

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

Python Package Index: https://pypi.python.org/pypi/nbsphinx/

License: MIT – see the file `LICENSE` for details.

Quick Start:

1. Install `nbsphinx`:

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

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!

[1] http://sphinx-doc.org/
[2] http://jupyter.org/

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

Usage

Installation

Install `nbsphinx` with `pip`:

```
python3 -m pip install nbsphinx --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`. Recent versions of Python already come with `pip` pre-installed. If you don't have it, you can install it manually[3].

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:

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

You'll most likely have this installed already.

Jupyter Kernel

If you want to execute your notebooks during the Sphinx build process (see *Executing Notebooks*), you need an appropriate Jupyter kernel[4] installed.

For example, if you use Python, you should install the `ipykernel` package:

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

Again, it's very likely that you have that installed already.

Sphinx Setup

In the directory with your notebook files, run this command (assuming you have Sphinx[5] installed already):

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

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

[3] https://pip.pypa.io/en/latest/installing/
[4] http://jupyter.readthedocs.io/en/latest/projects/kernels.html
[5] http://sphinx-doc.org/

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']
```

Once your `conf.py` is in place, edit the file named `index.rst` and add the file names of your notebooks (with or without the `.ipynb` extension) to the toctree[6] directive.

---

autosummary bug:

If you are using the `sphinx.ext.autosummary` Sphinx extension, there is a bug in Sphinx (below version 1.5)[7] which prevents notebooks from being parsed. As a work-around you can explicitly list all the files for which autosummary should be ran using the autosummary_generate[8] variable in `conf.py`. For example,

```
autosummary_generate = ['myfile1.rst', 'myfile2.rst']
```

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.

To create LaTeX output, use:

```
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[9] (see also this tutorial[10]).

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

---

[6] http://www.sphinx-doc.org/en/stable/markup/toctree.html
[7] https://github.com/sphinx-doc/sphinx/issues/2485
[8] http://www.sphinx-doc.org/en/stable/ext/autosummary.html#confval-autosummary_generate
[9] http://users.phys.psu.edu/~collins/software/latexmk-jcc/
[10] http://mg.readthedocs.io/latexmk.html

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

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[11]!

It can be installed with

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

You can start auto-building your files with

```
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:

```
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*). 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.

Automatic Creation of HTML and PDF output on readthedocs.org[12]

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 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.

If your new versions are not automatically built, go to the "Settings" of your Github repository, continue to "Integrations & services", and make sure that "ReadTheDocs" is listed and activated in the "Services" section. If not, use "Add service". There is probably a similar thing for Bitbucket and others.

Note:

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

---

[11] https://pypi.python.org/pypi/sphinx-autobuild
[12] https://readthedocs.org

Using `requirements.txt`

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

```
sphinx>=1.4
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!

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[13] and should contain something like this:

```
channels:
  - conda-forge
dependencies:
  - python>=3
  - sphinx>=1.4
  - pandoc
  - nbconvert
  - ipykernel
  - pip:
    - nbsphinx
```

   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 necessary for the `pandoc` package, which is not part of the default channel.

[13] http://conda.pydata.org/docs/using/envs.html

The currently pre-installed version of `pandoc` doesn't seem to convert HTML5 `<audio>` elements correctly. See nbsphinx issue #69[14] and readthedocs.org issue #2521[15].

Note:

Most of the "Advanced Settings" on readthedocs.org will be ignored if you have a `readthedocs.yml` file.

Warning:

If you have a very long repository name (or branch name), you might run into this quite obscure problem: 'placeholder too short'[16].

HTML Themes

The `nbsphinx` extension does *not* provide its own theme, you can use any of the available themes or create a custom one[17], if you feel like it.

The following (incomplete) list of themes contains several 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[18])
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)

Sphinx's Built-In Themes

- alabaster[19]: example[20], usage[21]
- `pyramid`: example[22], usage[23]
- `classic`: example[24], usage[25]
- `bizstyle`: example[26], usage[27]
- `haiku`: example[28], usage[29]
- `traditional`: example[30], usage[31]

[14] https://github.com/spatialaudio/nbsphinx/issues/69
[15] https://github.com/rtfd/readthedocs.org/issues/2521
[16] https://github.com/rtfd/readthedocs.org/issues/1902
[17] http://www.sphinx-doc.org/en/stable/theming.html#creating-themes
[18] http://www.sphinx-doc.org/en/latest/theming.html#builtin-themes
[19] https://alabaster.readthedocs.io/
[20] http://nbsphinx.readthedocs.io/en/alabaster-theme/
[21] https://github.com/spatialaudio/nbsphinx/compare/alabaster-theme%5E...alabaster-theme
[22] http://nbsphinx.readthedocs.io/en/pyramid-theme/
[23] https://github.com/spatialaudio/nbsphinx/compare/pyramid-theme%5E...pyramid-theme
[24] http://nbsphinx.readthedocs.io/en/classic-theme/
[25] https://github.com/spatialaudio/nbsphinx/compare/classic-theme%5E...classic-theme
[26] http://nbsphinx.readthedocs.io/en/bizstyle-theme/
[27] https://github.com/spatialaudio/nbsphinx/compare/bizstyle-theme%5E...bizstyle-theme
[28] http://nbsphinx.readthedocs.io/en/haiku-theme/
[29] https://github.com/spatialaudio/nbsphinx/compare/haiku-theme%5E...haiku-theme
[30] http://nbsphinx.readthedocs.io/en/traditional-theme/
[31] https://github.com/spatialaudio/nbsphinx/compare/traditional-theme%5E...traditional-theme

- `agogo`: example[32], usage[33]
- `nature`: example[34], usage[35]

3rd-Party Themes

- sphinx_rtd_theme[36]: example[37], usage[38]
- bootstrap[39]: example[40], usage[41]
- cloud[42]: example[43], usage[44]
- sphinx_py3doc_enhanced_theme[45]: example[46], usage[47]
- basicstrap[48]: example[49], usage[50]
- dotted[51]: example[52], usage[53]
- better[54]: example[55], usage[56]
- guzzle_sphinx_theme[57]: example[58], usage[59]
- julia[60]: example[61], usage[62]

If you know of another Sphinx theme that should be included here, please open an issue on Github[63].

Markdown Cells

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

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

- Red

[32] http://nbsphinx.readthedocs.io/en/agogo-theme/
[33] https://github.com/spatialaudio/nbsphinx/compare/agogo-theme%5E...agogo-theme
[34] http://nbsphinx.readthedocs.io/en/nature-theme/
[35] https://github.com/spatialaudio/nbsphinx/compare/nature-theme%5E...nature-theme
[36] https://github.com/snide/sphinx_rtd_theme
[37] http://nbsphinx.readthedocs.io/en/rtd-theme/
[38] https://github.com/spatialaudio/nbsphinx/compare/rtd-theme%5E...rtd-theme
[39] http://sphinx-bootstrap-theme.readthedocs.io/
[40] http://nbsphinx.readthedocs.io/en/bootstrap-theme/
[41] https://github.com/spatialaudio/nbsphinx/compare/bootstrap-theme%5E...bootstrap-theme
[42] https://pythonhosted.org/cloud_sptheme/
[43] http://nbsphinx.readthedocs.io/en/cloud-theme/
[44] https://github.com/spatialaudio/nbsphinx/compare/cloud-theme%5E...cloud-theme
[45] https://github.com/ionelmc/sphinx-py3doc-enhanced-theme
[46] http://nbsphinx.readthedocs.io/en/py3doc-enh-theme/
[47] https://github.com/spatialaudio/nbsphinx/compare/py3doc-enh-theme%5E...py3doc-enh-theme
[48] http://pythonhosted.org/sphinxjp.themes.basicstrap/
[49] http://nbsphinx.readthedocs.io/en/basicstrap-theme/
[50] https://github.com/spatialaudio/nbsphinx/compare/basicstrap-theme%5E...basicstrap-theme
[51] https://pythonhosted.org/sphinxjp.themes.dotted/
[52] http://nbsphinx.readthedocs.io/en/dotted-theme/
[53] https://github.com/spatialaudio/nbsphinx/compare/dotted-theme%5E...dotted-theme
[54] https://sphinx-better-theme.readthedocs.io/
[55] http://nbsphinx.readthedocs.io/en/better-theme/
[56] https://github.com/spatialaudio/nbsphinx/compare/better-theme%5E...better-theme
[57] https://github.com/guzzle/guzzle_sphinx_theme
[58] http://nbsphinx.readthedocs.io/en/guzzle-theme/
[59] https://github.com/spatialaudio/nbsphinx/compare/guzzle-theme%5E...guzzle-theme
[60] https://github.com/JuliaLang/JuliaDoc
[61] http://nbsphinx.readthedocs.io/en/julia-theme/
[62] https://github.com/spatialaudio/nbsphinx/compare/julia-theme%5E...julia-theme
[63] https://github.com/spatialaudio/nbsphinx/issues

- Green
- Blue

---

1. One
2. Two
3. Three

Equations

Equations can be formatted really nicely, either inline, like $e^{i\pi} = -1$, or on a separate line, like

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

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[64]:

> 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.

Code

We can also write code with nice syntax highlighting:

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

Tables

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

[64] http://pandoc.org/README.html#math

Images

PNG file (local):

SVG file (local):

PNG file (remote):

SVG file (remote):

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[65] 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[66] elements can be created like this:

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

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

Example:

The HTML video element is not supported!

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.

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[67]), such boxes can be created by using HTML `<div>` 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 `<div>` 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!

Links to Other Notebooks

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

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

This is how a link is created in Markdown:

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

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

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*.

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*.

This can be done, as expected, like this:

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

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

Links to files whose extension is in the configuration value source_suffix[68], will be converted to links to the generated HTML/LaTeX pages. Example: *A reStructuredText file*.

This was created with:

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

Links to sub-sections are not (yet?) possible.

Links to Local Files (HTML only)

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, no link is created.

Code Cells

Code, Output, Streams

An empty code cell:

```
In [1]:
```

[68] http://www.sphinx-doc.org/config.html#confval-source_suffix

Two empty lines:

```
In [1]:
```

Leading/trailing empty lines:

```
In [1]:

        # 2 empty lines before, 1 after
```

A simple output:

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

```
Out[2]: 42
```

The standard output stream:

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

```
Hello, world!
```

Normal output + standard output

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

```
Hello, world!
```

```
Out[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.

```
In [5]: import sys

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

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

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

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

Cell Magics

Cells can contain code in other languages by means of cell magics[69]:

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

```
1
2
3
```

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

Special Display Formats

See IPython example notebook[70].

TODO: tables? e.g. Pandas DataFrame?

```
In [7]: from IPython.display import display
```

Local Image Files

```
In [8]: from IPython.display import Image
        i = Image(filename='images/notebook_icon.png')
        i
```


```
In [9]: display(i)
```

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

For some reason this doesn't work with `Image(...)`:

```
In [10]: from IPython.display import SVG
         SVG(filename='images/python_logo.svg')
```

Image URLs

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

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

```
In [12]: Image(url='https://www.python.org/static/img/python-logo-large.png', embed=True)
```

In [13]:
```
Image(url='http://jupyter.org/assets/nav_logo.svg')
```

Out[13]:
```
<IPython.core.display.Image object>
```

In [14]:
```
Image(url='https://www.python.org/static/favicon.ico')
```

Out[14]:
```
<IPython.core.display.Image object>
```

In [15]:
```
Image(url='http://python.org/images/python-logo.gif')
```

Out[15]:
```
<IPython.core.display.Image object>
```

Math

In [16]:
```
from IPython.display import Math
eq = Math(r"\int_{-\infty}^\infty f(x) \delta(x - x_0) dx = f(x_0)")
eq
```

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

In [17]:
```
display(eq)
```

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

In [18]:
```
%%latex
\begin{equation}
\int_{-\infty}^\infty f(x) \delta(x - x_0) dx = f(x_0)
\end{equation}
```

$$\int_{-\infty}^{\infty} f(x)\delta(x - x_0)dx = f(x_0) \quad (2)$$

YouTube Videos

```
In [19]: from IPython.display import YouTubeVideo
         YouTubeVideo('WAikxUGbomY')
```

Arbitrary JavaScript Output (HTML only)

```
In [20]: %%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>
```

Note:

jQuery should be available, but using the readthedocs.org default theme, it's not. See the issue on Github[71]. Other Sphinx themes are not affected by this.

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.

[71] https://github.com/snide/sphinx_rtd_theme/issues/328

```
In [21]: display({
             'text/x-python': 'print("Hello, world!")',
             'text/x-haskell': 'main = putStrLn "Hello, world!"',
         }, raw=True)
```

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

ANSI Colors

The standard output and standard error streams may contain ANSI escape sequences[72] to change the text and background colors.

```
In [22]: print('BEWARE: \x1b[1;33;41mugly colors\x1b[m!', file=sys.stderr, flush=True)
         print('ABC\x1b[43mDEF\x1b[35mGHI\x1b[1mJKL\x1b[49mMNO\x1b[39mPQR\x1b[22mSTU')
```

```
BEWARE: ugly colors!
ABCDEFGHIJKLMNOPQRSTU
```

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.

```
In [23]: text = ' XYZ '
         formatstring = '\x1b[{}m' + text + '\x1b[m'

         print('    ' + '  ' * 6 + '    ' + '  ' * 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.

```
In [24]: formatstring = '\x1b[38;5;{0};48;5;{0}mX\x1b[1mX\x1b[m'
```

[72] https://en.wikipedia.org/wiki/ANSI_escape_code

```
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 91011121314151617181920212223242526272829303132333435
  0  X X X X X X X X
 16
 52
 88
124
160
196
232
```

You can even use 24-bit RGB colors:

```
In [25]: 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))
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

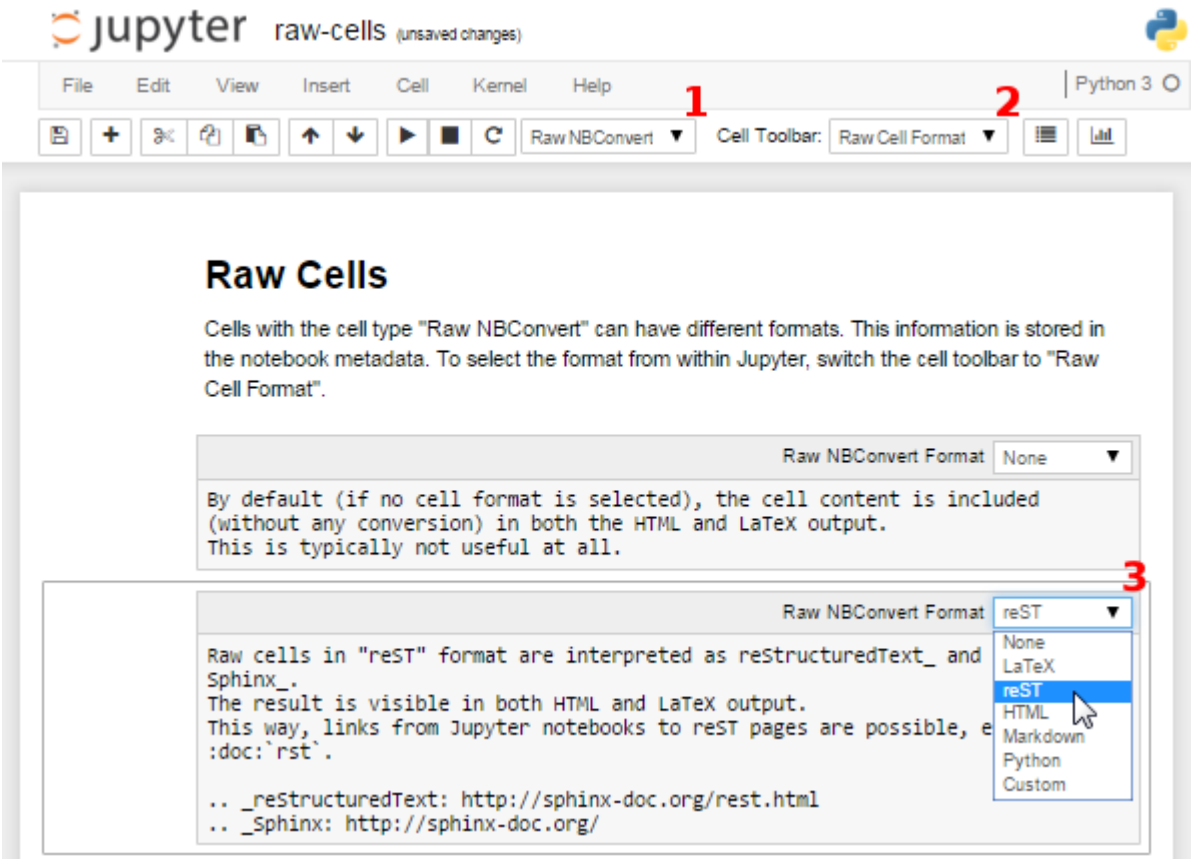
Raw Cells

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.

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



Available Raw Cell Formats

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

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."

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[73] format."

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[74] format."

[73] http://sphinx-doc.org/rest.html
[74] https://daringfireball.net/projects/markdown/

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.

LaTeX

Raw cells in "LaTeX" format are only visible in LaTeX output. **I'm** a *raw cell* LaTeX format.

Python

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

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:

```
In [2]: answer
```

```
Out[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.

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.

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.

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 have the time – and then just keep the output.

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

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

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

```
Out[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*.

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.

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

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

```
Out[4]: 42
```

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 basis – see *Ignoring Errors*.
2. Execute the notebook beforehand and save the results, like it's done in this example notebook:

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

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

ZeroDivisionError: division by zero
```

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:

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

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 three options:

1. Manually execute the notebook in question and save the results, see *the pre-executed example notebook*.
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
   },
   ```

This very notebook is an example for the last 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.

```
In [1]: nonsense
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-0377438312a9> in <module>()
----> 1 nonsense

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

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

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-b75601cc3487> in <module>()
----> 1 42 / 0

ZeroDivisionError: division by zero
```

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

```
  File "<ipython-input-3-788c64630141>", line 1
    print 'Hello, world!'
                        ^
SyntaxError: Missing parentheses in call to 'print'
```

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

```
  File "<ipython-input-4-07371befe33b>", line 1
    6 ~ 7
      ^
SyntaxError: invalid syntax
```

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

```
Out[5]: 42
```

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  # Time in seconds; use -1 for no timeout
```

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

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

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

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:

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

Warning:

There may be problems with images in output cells if your source directory contains symbolic links, see issue #49[75].

A link to a notebook in the parent directory: *link*.

A link to a local file: link.

A Sub-Section

This is just for testing inter-notebook links, see *this section*.

[75] https://github.com/spatialaudio/nbsphinx/issues/49

Using `toctree` In A Notebook

In Sphinx-based documentation, there is typically a file called `index.rst` which contains one or more toctree[76] directives. Those can be used to pull in further source files (which themselves can contain `toctree` directives).

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[77]. All options can be used – except `:glob:`, which can only be used in *rst files* and in *raw reST cells*.

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* 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.

[76] http://www.sphinx-doc.org/en/stable/markup/toctree.html
[77] http://www.sphinx-doc.org/markup/toctree.html

Yet Another Notebook

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

Normal reStructuredText Files

This is a normal RST file.

Note: Those still work!

Links to Notebooks

Links to notebooks can be easily created: *Notebooks in Sub-Directories* (the notebook title is used as link text). You can also use *an alternative text*.

The above links were created with:

```
:ref:`subdir/a-notebook-in-a-subdir.ipynb`
:ref:`an alternative text <subdir/a-notebook-in-a-subdir.ipynb>`
```

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

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:

- Spaces in the section title have to be replaced by hyphens!
- "../" is not allowed, you have to specify the full path even if the current source file is in a subdirectory!

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:

> **Warning:** You should probably not use nested boxes!

External Links

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

https://github.com/ipython/nbconvert/pull/35

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

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

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

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

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