nbsphinx

Release 0.2.1

Matthias Geier

January 05, 2016

Contents

`nbsphinx` is a Sphinx extension that provides a source parser for `*.ipynb` files. Custom Sphinx directives are used to show Jupyter Notebook 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.rtfd.org/

Code: http://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` with pip:

```
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 (without the `.ipynb` extension) to the `toctree` directive.
4. Run Sphinx!

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

1 Usage

Install `nbsphinx` with pip:

```
pip install nbsphinx --user
```

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

```
pip uninstall nbsphinx
```

1.1 Sphinx Setup

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

```
sphinx-quickstart
```

Answer the questions that appear on the screen. In case of doubt, just press the `<Return>` key 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 `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 `index.rst` and add the file names of your notebooks (without the `.ipynb` extension) to the `toctree` directive.

1.2 Running Sphinx

To create the HTML pages, use this command:

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

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

```
sphinx-build <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:

```
sphinx-build . _build -j4
```

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

To create LaTeX output, use:

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

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.

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

This is very easy!

1. Create an account on https://readthedocs.org/ and add your Github/Bitbucket repository (or any publicly available Git/Subversion/Mercurial/Bazaar repository).
2. Create a file named `requirements.txt` (or whatever name you wish) in your repository containing the required pip packages:

```
    nbsphinx
    ipykernel
```

3. In the "Advanced Settings" on readthedocs.org, specify your `requirements.txt` file (or however you called it) in the box labelled "Requirements file". Kinda obvious, isn't it?
4. 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!
5. Make sure that in the "Settings" of your Github repository, under "Webhooks & services", "ReadTheDocs" is listed and activated in the "Services" section. If not, use "Add service". There is probably a similar thing for Bitbucket.
6. Done!

After that, 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.

1.4 HTML Themes

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

Here are a few examples how the `nbsphinx` input and output cells look like in different themes:

http://nbsphinx.readthedocs.org/en/readthedocs-theme/

http://nbsphinx.readthedocs.org/en/alabaster-theme/

http://nbsphinx.readthedocs.org/en/bootstrap-theme/

http://nbsphinx.readthedocs.org/en/cloud-theme/

http://nbsphinx.readthedocs.org/en/py3doc-enhanced-theme/

2 An Example Notebook

This notebook is meant for testing conversion to other formats.

It contains Markdown cells and code cells with different kinds of outputs.

2.1 Markdown

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

Equations

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

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

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

Images

Jupyter notebook icon (local):

Python logo (local):

Jupyter logo (remote):

Python logo (remote):

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).

This is how a link is created in Markdown:

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

2.2 Code Cells

An empty code cell:

```
In [1]:
```

A cell with no output:

```
In [1]: None
```

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 logging
        logging.warning('I am a warning and I will appear on the standard error stream')
        print('I will appear on the standard output stream')
        'I am the "normal" output'
```

```
WARNING:root:I am a warning and I will appear on the standard error stream
```

```
I will appear on the standard output stream
```

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

2.3 Special Display Formats

See IPython example notebook.

TODO: tables? e.g. Pandas DataFrame?

```
In [6]: from IPython.display import display, Image, SVG, Math, YouTubeVideo
```

Local Image Files

```
In [7]: i = Image(filename='images/notebook_icon.png')
        i
```

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

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

```
In [9]: SVG(filename='images/python_logo.svg')
```

Image URLs

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

In [11]: Image(url='https://www.python.org/static/img/python-logo-large.png', embed=True
In [12]: Image(url='http://jupyter.org/assets/nav_logo.svg')
```

```
In [13]: Image(url='https://www.python.org/static/favicon.ico')
```

```
In [14]: Image(url='http://python.org/images/python-logo.gif')
```

Math

```
In [15]: 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 [16]: display(eq)
```

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

```
In [17]: %%latex
         \begin{equation}
         \int_{-\infty}^\infty f(x) \delta(x - x_0) dx = f(x_0)
         \end{equation}
```

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

```
In [18]: YouTubeVideo('iV2ViNJFZC8')
```

2.4 Raw Cells

Cells with the cell type “Raw NBConvert” can have different formats. This information is stored in the notebook metadata. To select the format from within Jupyter, switch the cell toolbar to “Raw Cell Format”. 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. Raw cells in “reST” format are interpreted as reStructuredText and parsed by Sphinx. The result is visible in both HTML and LaTeX output. This way, links from Jupyter notebooks to reST pages are possible, e.g. A Normal reStructuredText File.

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*. Raw cells in “LaTeX” format are only included in LaTeX output (without any conversion).

Raw cells in “HTML” format are only included in HTML output. This might not be *very useful*, since raw HTML code is also allowed within “normal” Markdown cells. Raw cells in “Python” format are not shown at all (nor acted upon in any way).

3 A Pre-Executed Notebook

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.

This can be useful for the following use cases.

3.1 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 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
```

3.2 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.

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

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

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

3.3 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 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
```

4 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 now), you can add this to the notebook's JSON metadata:

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

This very notebook is an example for this behavior. 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, 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
```

5 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.

6 A Notebook in a Sub-Directory

Let's see if links to local images work:

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

A link to a notebook in the parent directory: link

7 A Normal reStructuredText File

This is a normal RST file.

Note: Those still work!

7.1 Links to Notebooks

Links to notebooks can be easily created: A Notebook in a Sub-Directory (the notebook title is used as link text). You can also use an alternative text. Relative links to notebooks in subdirectories are possible.

The above links were created with (note that the `.ipynb` file extension has to be stripped):

```
:doc:`subdir/another`
:doc:`an alternative text <subdir/another>`
```

7.2 Sphinx Directives for Jupyter Notebook Cells

For comparison, this is a "normal" Sphinx code block using `ipython3` syntax highlighting:

```
%file helloworld.py
#!/usr/bin/env python3
print('Hello, world!')
```

The nbsphinx extension provides custom directives to show notebook cells:

```
In [42]: 6 * 7
```

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

This was created with

```
.. nbinput:: ipython3
    :execution-count: 42

    6 * 7

.. nboutput::
    :execution-count: 42

    42
```

8 External Links

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

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

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

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

https://mail.scipy.org/pipermail/ipython-user/2013-December/013490.html

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

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

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

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

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

https://github.com/jupyter/nbconvert/issues/47

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

http://sphinx-doc.org/extdev/

https://github.com/sphinx-doc/sphinx/issues/1907