nbsphinx

Release 0.2.4

Matthias Geier

February 13, 2016

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.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[3]:

```
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[4]:

```
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[5] installed already):

```
sphinx-quickstart
```

[1] http://sphinx-doc.org/
[2] http://jupyter.org/
[3] https://pip.pypa.io/en/latest/installing/
[4] https://pip.pypa.io/en/latest/installing/
[5] http://sphinx-doc.org/

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 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 (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 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

2.1 Equations

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

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

2.2 Code

We can also write code with nice syntax highlighting:

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

2.3 Tables

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

2.4 Images

PNG file (local):

SVG file (local):

PNG file (remote):

SVG file (remote):

2.5 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
```

Links to sub-sections are also possible, e.g. *this subsection*.

This link was created with:

```
[this subsection](subdir/another.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)
```

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

Links to files whose extension is in the configuration value source_suffix[6], will be converted to links to the generated HTML/LaTeX pages. Example: A reStructuredText file.

This was created with:

```
[A reStructuredText file](rst.rst)
```

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

2.7 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.

3 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
```

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

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"
```

3.1 Special Display Formats

See IPython example notebook[7].

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
```

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

```
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}
```

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

YouTube Videos

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

3.2 ANSI Colors

The standard output and standard error streams may contain ANSI escape sequences[8] to change the text and background colors.

```
In [19]: 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. Note that Jupyter does not switch to a brighter color for bold text.

```
In [20]: 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  XYZ
 1;30  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
   31  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
 1;31  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
   32  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
 1;32  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
   33  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
 1;33  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
   34  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
 1;34  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
   35  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
 1;35  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
   36  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
 1;36  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ  XYZ
   37  XYZ  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 [21]: 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 91011121314151617181920212223242526272829303132333435
  0 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 16 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

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

```
 52 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 88 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
124 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
160 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
196 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
232 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

You can even use 24-bit RGB colors:

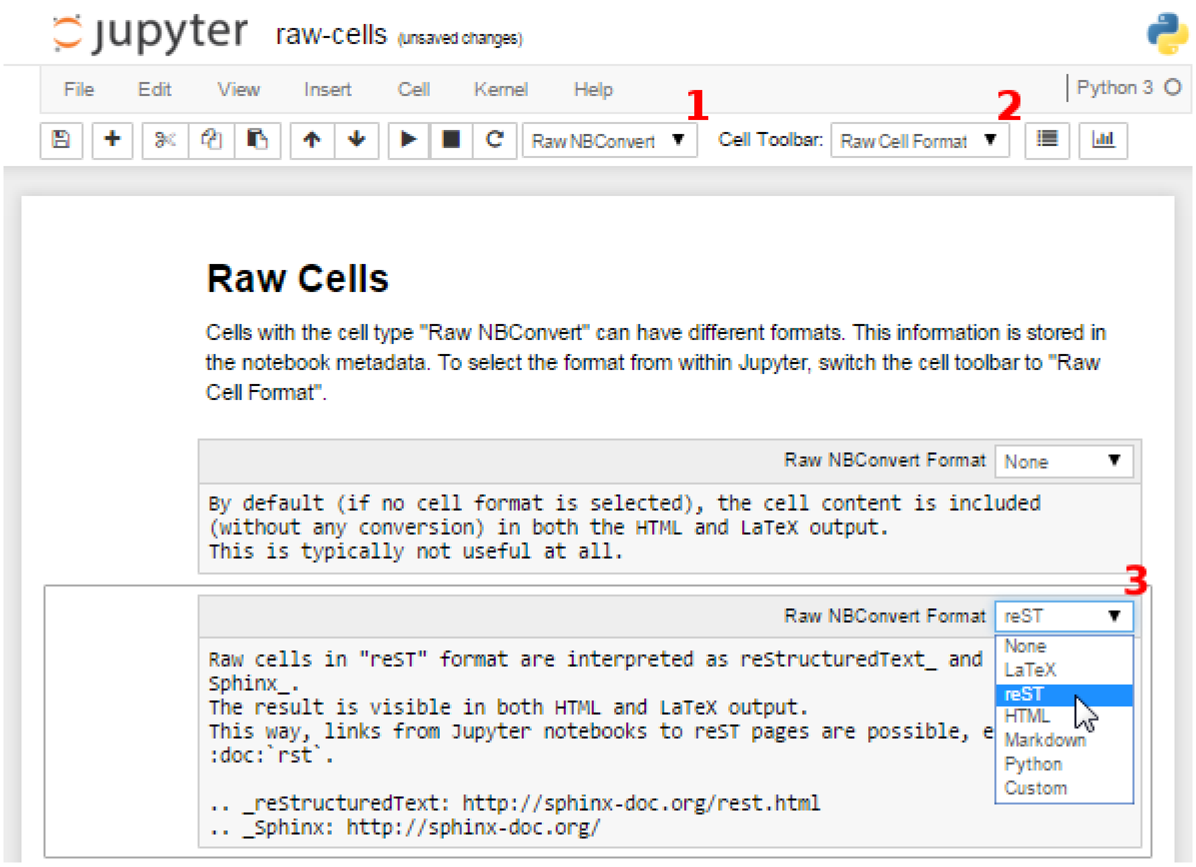
```
In [22]: start = 255, 0, 0
         end = 0, 0, 255
         length = 79
         out = []

         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
```

4 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. 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


jupyter raw-cells (unsaved changes)
File Edit View Insert Cell Kernel Help
Python 3
1
Raw NBConvert
Cell Toolbar: Raw Cell Format
2
3
None
LaTeX
reST
HTML
Markdown
Python
Custom

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".

Raw NBConvert Format: 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.
```

Raw NBConvert Format: reST

```
Raw cells in "reST" format are interpreted as reStructuredText_ and
Sphinx_.
The result is visible in both HTML and LaTeX output.
This way, links from Jupyter notebooks to reST pages are possible, e
:doc:`rst`.

.. _reStructuredText: http://sphinx-doc.org/rest.html
.. _Sphinx: http://sphinx-doc.org/
```

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

4.1 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."

4.2 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[9] format.”

4.3 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[10] format.”

4.4 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.

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

4.5 Latex

Raw cells in “LaTeX” format are only visible in LaTeX output. **I’m** a *raw cell* LaTeX format.

4.6 Python

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

5 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.

5.1 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
```

5.2 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
```

5.3 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
```

6 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
```

7 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.

8 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.

A link to a local file: link.

8.1 A Sub-Section

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

9 Using `toctree` In A Notebook

In Sphinx-based documentation, there is typically a file called index.rst which contains one or more `toctree` 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 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.

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[11]. 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.

9.1 Yet Another Notebook

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

10 A Normal reStructuredText File

This is a normal RST file.

Note: Those still work!

10.1 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*.

The above links were created with:

```
:ref:`subdir/another.ipynb`
:ref:`an alternative text <subdir/another.ipynb>`
```

[11] http://www.sphinx-doc.org/markup/toctree.html

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/another.ipynb#A-Sub-Section`
:ref:`alternative text <subdir/another.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!

10.2 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
```

11 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.org/

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.org/

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