nbconvert Documentation

Release 6.0.1

Jupyter Development Team

Sep 10, 2020

USER DOCUMENTATION

1 Installation **3**
- 1.1 Supported Python versions . . . 3
- 1.2 Installing nbconvert . . . 3
- 1.3 Installing Pandoc . . . 3
- 1.4 Installing TeX . . . 4
- 1.5 Installing Chromium . . . 4

2 Using as a command line tool **5**
- 2.1 Default output format - HTML . . . 5
- 2.2 Supported output formats . . . 5
- 2.3 Converting multiple notebooks . . . 9

3 Using nbconvert as a library **11**
- 3.1 Quick overview . . . 11
- 3.2 Extracting Figures using the RST Exporter . . . 13
- 3.3 Extracting Figures using the HTML Exporter . . . 15
- 3.4 Custom Preprocessors . . . 17
- 3.5 Example . . . 17
- 3.6 Programmatically creating templates . . . 18
- 3.7 Real World Uses . . . 18

4 LaTeX citations **19**

5 Removing cells, inputs, or outputs **21**
- 5.1 Removing pieces of cells using cell tags . . . 21
- 5.2 Removing cells using Regular Expressions on cell content . . . 21

6 Executing notebooks **23**
- 6.1 Executing notebooks from the command line . . . 23
- 6.2 Executing notebooks using the Python API interface . . . 23
- 6.3 Execution arguments (traitlets) . . . 24
- 6.4 Handling errors and exceptions . . . 24
- 6.5 Widget state . . . 25

7 Configuration options **27**
- 7.1 CLI Flags and Aliases . . . 27
- 7.2 App Options . . . 28
- 7.3 Exporter Options . . . 31
- 7.4 Writer Options . . . 46
- 7.5 Preprocessor Options . . . 49
- 7.6 Postprocessor Options . . . 55

7.7 Other Options . . . 56

8 Creating Custom Templates for nbconvert **59**
8.1 Selecting a template . . . 59
8.2 Where are nbconvert templates installed? . . . 59
8.3 The content of nbconvert templates . . . 60

9 Customizing exporters **63**
9.1 Extending the built-in format exporters . . . 63
9.2 Registering a custom exporter as an entry point . . . 63
9.3 Using a custom exporter without entrypoints . . . 64

10 Parameters controlled by an external exporter **65**

11 Writing a custom `Exporter` **67**

12 Customizing Syntax Highlighting **71**
12.1 Using Builtin styles . . . 71
12.2 Making your own styles . . . 72

13 Architecture of nbconvert **73**
13.1 A detailed pipeline exploration . . . 73
13.2 Classes . . . 74

14 Python API for working with nbconvert **77**
14.1 NbConvertApp . . . 77
14.2 Exporters . . . 78
14.3 Preprocessors . . . 83
14.4 Filters . . . 86
14.5 Writers . . . 89
14.6 Postprocessors . . . 89

15 Making an `nbconvert` release **91**
15.1 Assign all merged PRs to milestones . . . 91
15.2 Gather all PRs related to milestone . . . 91
15.3 Manually categorize tickets . . . 91
15.4 Collect major changes . . . 91
15.5 Update docs/source/changelog.rst . . . 92
15.6 Check installed tools . . . 92
15.7 Clean the repository . . . 92
15.8 Create the release . . . 92
15.9 Release the new version . . . 93
15.10 Return to development state . . . 93
15.11 Email googlegroup with update letter . . . 93

16 Changes in nbconvert **95**
16.1 6.0.0 . . . 95
16.2 5.6.1 . . . 97
16.3 5.6 . . . 98
16.4 5.5 . . . 100
16.5 5.4.1 . . . 103
16.6 5.4 . . . 104
16.7 5.3.1 . . . 108
16.8 5.3 . . . 108
16.9 5.2.1 . . . 109

16.10 5.1.1 . . . 112
16.11 5.1 . . . 112
16.12 5.0 . . . 112
16.13 4.3 . . . 113
16.14 4.2 . . . 113
16.15 4.1 . . . 113
16.16 4.0 . . . 113

17 Indices and tables **115**

Python Module Index **117**

Index **119**

Using `nbconvert` enables:

- **presentation** of information in familiar formats, such as PDF.
- **publishing** of research using LaTeX and opens the door for embedding notebooks in papers.
- **collaboration** with others who may not use the notebook in their work.
- **sharing** contents with many people via the web using HTML.

Overall, notebook conversion and the `nbconvert` tool give scientists and researchers the flexibility to deliver information in a timely way across different formats.

Primarily, the `nbconvert` tool allows you to convert a Jupyter `.ipynb` notebook document file into another static format including HTML, LaTeX, PDF, Markdown, reStructuredText, and more. `nbconvert` can also add productivity to your workflow when used to execute notebooks programmatically.

If used as a Python library (`import nbconvert`), `nbconvert` adds notebook conversion within a project. For example, `nbconvert` is used to implement the "Download as" feature within the Jupyter Notebook web application. When used as a command line tool (invoked as `jupyter nbconvert ...`), users can conveniently convert just one or a batch of notebook files to another format.

Contents:

CHAPTER

ONE

INSTALLATION

See also:

Installing Jupyter Nbconvert is part of the Jupyter ecosystem.

1.1 Supported Python versions

Currently Python 3.6-3.8 is supported and tested by nbconvert.

However, nbconvert 6.0 provides limited support for Python 3.6. nbconvert 6.1 will drop support for Python 3.6. Limited support means we will test and run CI on Python 3.6.12 or higher. Issues that are found only affecting Python 3.6 are not guaranteed to be fixed. We recommend all users of nbconvert use Python 3.7 and higher.

1.2 Installing nbconvert

Nbconvert is packaged for both pip and conda, so you can install it with:

```
pip install nbconvert

# OR

conda install nbconvert
```

The Miniconda and Miniforge distributions both provide a minimal conda installation.

Important: To unlock its full capabilities, nbconvert requires Pandoc, TeX (specifically, XeLaTeX) and Pyppeteer. These must be installed separately.

1.3 Installing Pandoc

For converting markdown to formats other than HTML, nbconvert uses Pandoc (1.12.1 or later).

To install pandoc on Linux, you can generally use your package manager:

```
sudo apt-get install pandoc
```

On other platforms, you can get pandoc from their website.

1.4 Installing TeX

For converting notebooks to PDF (with `--to pdf`), nbconvert makes use of LaTeX and the XeTeX as the rendering engine.

New in version 5.0: We use XeTeX as the rendering engine rather than pdfTeX (as in earlier versions). XeTeX can access fonts through native operating system libraries, it has better support for OpenType formatted fonts and Unicode characters.

To install a complete TeX environment (including XeLaTeX and the necessary supporting packages) by hand can be tricky. Fortunately, there are packages that make this much easier. These packages are specific to different operating systems:

- Linux: TeX Live
  - E.g. on Debian or Ubuntu:

```
sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-generic-
↪recommended
```

- macOS (OS X): MacTeX.
- Windows: MikTex

Because nbconvert depends on packages and fonts included in standard TeX distributions, if you do not have a complete installation, you may not be able to use nbconvert's standard tooling to convert notebooks to PDF.

1.5 Installing Chromium

For converting notebooks to PDF with `--to webpdf`, nbconvert requires the Pyppeteer Chromium automation library.

Pyppeteer makes use of a specific version of Chromium. If it does not find a suitable installation of the web browser, it can automatically download it if the `--allow-chromium-download` flag is passed to the command line.

To install a suitable version of pyppeteer, you can pip install `nbconvert[webpdf]`.

1.5.1 PDF conversion on a limited TeX environment

If you are only able to install a limited TeX environment, there are two main routes you could take to convert to PDF:

1. **Using TeX by hand**
   a. You could convert to `.tex` directly; this requires Pandoc.
   b. edit the file to accord with your local environment
   c. run `xelatex` directly.
2. **Custom exporter**
   a. You could write a *custom exporter* that takes your system's limitations into account.

CHAPTER

TWO

USING AS A COMMAND LINE TOOL

The command-line syntax to run the `nbconvert` script is:

```
$ jupyter nbconvert --to FORMAT notebook.ipynb
```

This will convert the Jupyter notebook file `notebook.ipynb` into the output format given by the `FORMAT` string.

2.1 Default output format - HTML

The default output format is HTML, for which the `--to` argument may be omitted:

```
$ jupyter nbconvert notebook.ipynb
```

2.2 Supported output formats

The currently supported output formats are:

- *HTML*,
- *LaTeX*,
- *PDF*,
- *WebPDF*,
- *Reveal.js HTML slideshow*,
- *Markdown*,
- *Ascii*,
- *reStructuredText*,
- *executable script*,
- *notebook*.

Jupyter also provides a few templates for output formats. These can be specified via an additional `--template` argument and are listed in the sections below.

2.2.1 HTML

- `--to html`
  - `--template full` (default)

    A full static HTML render of the notebook. This looks very similar to the interactive view.
  - `--template basic`

    Simplified HTML, useful for embedding in webpages, blogs, etc. This excludes HTML headers.

2.2.2 LaTeX

- `--to latex`

  Latex export. This generates `NOTEBOOK_NAME.tex` file, ready for export. Images are output as .png files in a folder.
  - `--template article` (default)

    Latex article, derived from Sphinx's howto template.
  - `--template report`

    Latex report, providing a table of contents and chapters.

Note: nbconvert uses pandoc to convert between various markup languages, so pandoc is a dependency when converting to latex or reStructuredText.

2.2.3 PDF

- `--to pdf`

  Generates a PDF via latex. Supports the same templates as `--to latex`.

2.2.4 WebPDF

- `--to webpdf`

  Generates a PDF by first rendering to HTML, rendering the HTML Chromium headless, and exporting to PDF. This exporter supports the same templates as `--to html`.

  The webpdf exporter requires the `pyppeteer` Chromium automation library, which can be installed via `nbconvert[webpdf]`.

2.2.5 Reveal.js HTML slideshow

- `--to slides`

  This generates a Reveal.js HTML slideshow.

Running this slideshow requires a copy of reveal.js (version 3.x).

By default, this will include a script tag in the html that will directly load reveal.js from a public CDN.

This means that if you include your slides on a webpage, they should work as expected. However, some features (specifically, speaker notes & timers) will not work on website because they require access to a local copy of reveal.js.

Speaker notes require a local copy of reveal.js. Then, you need to tell `nbconvert` how to find that local copy.

Timers only work if you already have speaker notes, but also require a local https server. You can read more about this in *ServePostProcessorExample*.

To make this clearer, let's look at an example of how to get speaker notes working with a local copy of reveal.js: *SlidesWithNotesExample*.

Note: In order to designate a mapping from notebook cells to Reveal.js slides, from within the Jupyter notebook, select menu item View –> Cell Toolbar –> Slideshow. That will reveal a drop-down menu on the upper-right of each cell. From it, one may choose from "Slide," "Sub-Slide", "Fragment", "Skip", and "Notes." On conversion, cells designated as "skip" will not be included, "notes" will be included only in presenter notes, etc.

Example: creating slides w/ speaker notes

Let's suppose you have a notebook `your_talk.ipynb` that you want to convert to slides. For this example, we'll assume that you are working in the same directory as the notebook you want to convert (i.e., when you run `ls .`, `your_talk.ipynb` shows up amongst the list of files).

First, we need a copy of reveal.js in the same directory as your slides. One way to do this is to use the following commands in your terminal:

```
git clone https://github.com/hakimel/reveal.js.git
cd reveal.js
git checkout 3.5.0
cd ..
```

Then we need to tell nbconvert to point to this local copy. To do that we use the `--reveal-prefix` command line flag to point to the local copy.

```
jupyter nbconvert your_talk.ipynb --to slides --reveal-prefix reveal.js
```

This will create file `your_talk.slides.html`, which you should be able to access with `open your_talk.slides.html`. To access the speaker notes, press `s` after the slides load and they should open in a new window.

Note: This does not enable slides that run completely offline. While you have a local copy of reveal.js, by default, the slides need to access mathjax, require, and jquery via a public CDN. Addressing this use case is an open issue and PRs are always encouraged.

Serving slides with an https server: `--post serve`

Once you have speaker notes working you may notice that your timers don't work. Timers require a bit more infrastructure; you need to serve your local copy of reveal.js from a local https server.

Fortunately, `nbconvert` makes this fairly straightforward through the use of the `ServePostProcessor`. To activate this server, we append the command line flag `--post serve` to our call to nbconvert.

```
jupyter nbconvert your_talk.ipynb --to slides --reveal-prefix reveal.js --post serve
```

This will run the server, which will occupy the terminal that you ran the command in until you stop it. You can stop the server by pressing `ctrl C` twice.

2.2.6 Markdown

- `--to markdown`

  Simple markdown output. Markdown cells are unaffected, and code cells indented 4 spaces. Images are output as .png files in a folder.

2.2.7 Ascii

- `--to asciidoc`

  Ascii output. Images are output as .png files in a folder.

2.2.8 reStructuredText

- `--to rst`

  Basic reStructuredText output. Useful as a starting point for embedding notebooks in Sphinx docs. Images are output as .png files in a folder.

  Note: nbconvert uses pandoc to convert between various markup languages, so pandoc is a dependency when converting to LaTeX or reStructuredText.

2.2.9 Executable script

- `--to script`

  Convert a notebook to an executable script. This is the simplest way to get a Python (or other language, depending on the kernel) script out of a notebook. If there were any magics in an Jupyter notebook, this may only be executable from a Jupyter session.

  For example, to convert a Julia notebook to a Julia executable script:

  ```
  jupyter nbconvert --to script my_julia_notebook.ipynb
  ```

2.2.10 Notebook and preprocessors

- `--to notebook`

  New in version 3.0.

  This doesn't convert a notebook to a different format *per se*, instead it allows the running of nbconvert preprocessors on a notebook, and/or conversion to other notebook formats. For example:

```
jupyter nbconvert --to notebook --execute mynotebook.ipynb
```

This will open the notebook, execute it, capture new output, and save the result in `mynotebook.nbconvert.ipynb`. Specifying `--inplace` will overwrite the input file instead of writing a new file. By default, `nbconvert` will abort conversion if any exceptions occur during execution of a cell. If you specify `--allow-errors` (in addition to the `--execute` flag) then conversion will continue and the output from any exception will be included in the cell output.

The following command:

```
jupyter nbconvert --to notebook --nbformat 3 mynotebook
```

will create a copy of `mynotebook.ipynb` in `mynotebook.v3.ipynb` in version 3 of the notebook format.

If you want to convert a notebook in-place, you can specify the output file to be the same as the input file:

```
jupyter nbconvert --to notebook mynb --output mynb
```

Be careful with that, since it will replace the input file.

Note: nbconvert uses pandoc to convert between various markup languages, so pandoc is a dependency when converting to latex or reStructuredText.

The output file created by `nbconvert` will have the same base name as the notebook and will be placed in the current working directory. Any supporting files (graphics, etc) will be placed in a new directory with the same base name as the notebook, suffixed with `_files`:

```
$ jupyter nbconvert notebook.ipynb
$ ls
notebook.ipynb   notebook.html    notebook_files/
```

For simple single-file output, such as html, markdown, etc., the output may be sent to standard output with:

```
$ jupyter nbconvert --to markdown notebook.ipynb --stdout
```

2.3 Converting multiple notebooks

Multiple notebooks can be specified from the command line:

```
$ jupyter nbconvert notebook*.ipynb
$ jupyter nbconvert notebook1.ipynb notebook2.ipynb
```

or via a list in a configuration file, say `mycfg.py`, containing the text:

```
c = get_config()
c.NbConvertApp.notebooks = ["notebook1.ipynb", "notebook2.ipynb"]
```

and using the command:

```
$ jupyter nbconvert --config mycfg.py
```

CHAPTER

THREE

USING NBCONVERT AS A LIBRARY

In this notebook, you will be introduced to the programmatic API of nbconvert and how it can be used in various contexts.

A great blog post by @jakevdp will be used to demonstrate. This notebook will not focus on using the command line tool. The attentive reader will point-out that no data is read from or written to disk during the conversion process. This is because nbconvert has been designed to work in memory so that it works well in a database or web-based environment too.

3.1 Quick overview

Credit: Jonathan Frederic (@jdfreder on github)

The main principle of nbconvert is to instantiate an `Exporter` that controls the pipeline through which notebooks are converted.

First, download @jakevdp's notebook (if you do not have `requests`, install it by running `pip install requests`, or if you don't have pip installed, you can find it on PYPI):

[1]:
```
from urllib.request import urlopen

url = 'http://jakevdp.github.com/downloads/notebooks/XKCD_plots.ipynb'
response = urlopen(url).read().decode()
response[0:60] + ' ...'
```

[1]:
```
'{\n "cells": [\n  {\n   "cell_type": "markdown",\n   "metadata": ...'
```

The response is a JSON string which represents a Jupyter notebook.

Next, we will read the response using nbformat. Doing this will guarantee that the notebook structure is valid. Note that the in-memory format and on disk format are slightly different. In particual, on disk, multiline strings might be split into a list of strings.

[2]:
```
import nbformat
jake_notebook = nbformat.reads(response, as_version=4)
jake_notebook.cells[0]
```

[2]:
```
{'cell_type': 'markdown',
 'metadata': {},
 'source': '# XKCD plots in Matplotlib'}
```

The nbformat API returns a special type of dictionary. For this example, you don't need to worry about the details of the structure (if you are interested, please see the nbformat documentation).

The nbconvert API exposes some basic exporters for common formats and defaults. You will start by using one of them. First, you will import one of these exporters (specifically, the HTML exporter), then instantiate it using most of the defaults, and then you will use it to process the notebook we downloaded earlier.

[3]:
```
from traitlets.config import Config

# 1. Import the exporter
from nbconvert import HTMLExporter

# 2. Instantiate the exporter. We use the `classic` template for now; we'll get into␣
↪more details
# later about how to customize the exporter further.
html_exporter = HTMLExporter()
html_exporter.template_name = 'classic'

# 3. Process the notebook we loaded earlier
(body, resources) = html_exporter.from_notebook_node(jake_notebook)
```

The exporter returns a tuple containing the source of the converted notebook, as well as a resources dict. In this case, the source is just raw HTML:

[4]:
```
print(body[:400] + '...')
```

```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Notebook</title><script src="https://cdnjs.cloudflare.com/ajax/libs/require.js/
↪2.1.10/require.min.js"></script>



<style type="text/css">
    .highlight .hll { background-color: var(--jp-cell-editor-active-background) }
.highlight  { background: var(--jp-cell-edito...
```

If you understand HTML, you'll notice that some common tags are omitted, like the `body` tag. Those tags are included in the default `HtmlExporter`, which is what would have been constructed if we had not modified the `template_file`.

The resource dict contains (among many things) the extracted `.png`, `.jpg`, etc. from the notebook when applicable. The basic HTML exporter leaves the figures as embedded base64, but you can configure it to extract the figures. So for now, the resource dict should be mostly empty, except for a key containing CSS and a few others whose content will be obvious:

[5]:
```
print("Resources:", resources.keys())
print("Metadata:", resources['metadata'].keys())
print("Inlining:", resources['inlining'].keys())
print("Extension:", resources['output_extension'])
```

```
Resources: dict_keys(['metadata', 'output_extension', 'deprecated', 'theme', 'include_
↪css', 'include_js', 'include_url', 'require_js_url', 'jquery_url', 'inlining', 'raw_
↪mimetypes', 'global_content_filter'])
Metadata: dict_keys(['name'])
Inlining: dict_keys(['css'])
Extension: .html
```

`Exporters` are stateless, so you won't be able to extract any useful information beyond their configuration. You can re-use an exporter instance to convert another notebook. In addition to the `from_notebook_node` used above, each exporter exposes `from_file` and `from_filename` methods.

3.2 Extracting Figures using the RST Exporter

When exporting, you may want to extract the base64 encoded figures as files. While the HTML exporter does not do this by default, the `RstExporter` does:

[6]:
```
# Import the RST exproter
from nbconvert import RSTExporter
# Instantiate it
rst_exporter = RSTExporter()
# Convert the notebook to RST format
(body, resources) = rst_exporter.from_notebook_node(jake_notebook)

print(body[:970] + '...')
print('[.....]')
print(body[800:1200] + '...')
```

```
XKCD plots in Matplotlib
========================

This notebook originally appeared as a blog post at `Pythonic
Perambulations <http://jakevdp.github.com/blog/2012/10/07/xkcd-style-plots-in-
↪matplotlib/>`__
by Jake Vanderplas.

.. raw:: html

   <!-- PELICAN_BEGIN_SUMMARY -->

*Update: the matplotlib pull request has been merged! See* `This
post <http://jakevdp.github.io/blog/2013/07/10/XKCD-plots-in-matplotlib/>`__
*for a description of the XKCD functionality now built-in to
matplotlib!*

One of the problems I've had with typical matplotlib figures is that
everything in them is so precise, so perfect. For an example of what I
mean, take a look at this figure:

.. code:: ipython3

    from IPython.display import Image
    Image('http://jakevdp.github.com/figures/xkcd_version.png')




.. image:: output_3_0.png



Sometimes when showing schematic plots, this is the type of figure I
want to display. But drawing it by hand is a pain: I'd rather just use
matpl...
```

(continues on next page)

(continued from previous page)

```
[...]
mage:: output_3_0.png



Sometimes when showing schematic plots, this is the type of figure I
want to display. But drawing it by hand is a pain: I'd rather just use
matplotlib. The problem is, matplotlib is a bit too precise. Attempting
to duplicate this figure in matplotlib leads to something like this:

.. code:: ipython3

    Image('http://jakevdp.github.com/figures/mpl_version.png')




.. ima...
```

Notice that base64 images are not embedded, but instead there are filename-like strings, such as `output_3_0.png`. The strings actually are (configurable) keys that map to the binary data in the resources dict.

Note, if you write an RST Plugin, you are responsible for writing all the files to the disk (or uploading, etc...) in the right location. Of course, the naming scheme is configurable.

As an exercise, this notebook will show you how to get one of those images. First, take a look at the `'outputs'` of the returned resources dictionary. This is a dictionary that contains a key for each extracted resource, with values corresponding to the actual base64 encoding:

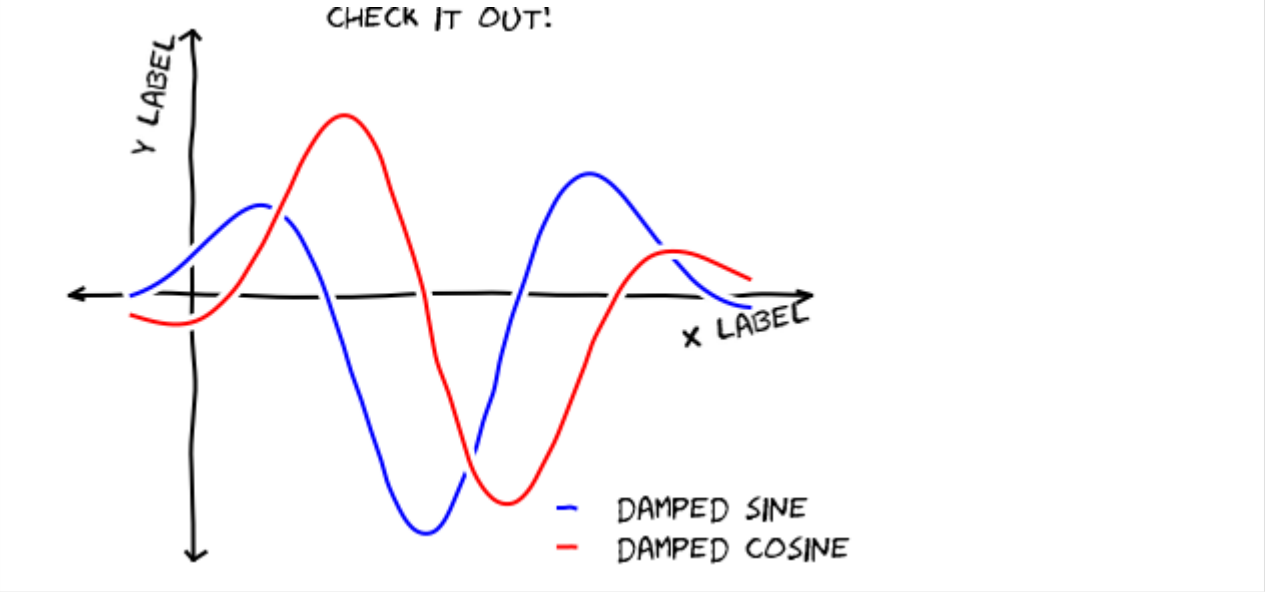
[7]:
```
sorted(resources['outputs'].keys())
```

[7]:
```
['output_13_1.png',
 'output_16_0.png',
 'output_18_1.png',
 'output_3_0.png',
 'output_5_0.png']
```

In this case, there are 5 extracted binary figures, all `pngs`. We can use the Image display object to actually display one of the images:

[8]:
```
from IPython.display import Image
Image(data=resources['outputs']['output_3_0.png'], format='png')
```

[8]:



Note that this image is being rendered without ever reading or writing to the disk.

3.3 Extracting Figures using the HTML Exporter

As mentioned above, by default, the HTML exporter does not extract images – it just leaves them as inline base64 encodings. However, this is not always what you might want. For example, here is a use case from @jakevdp:

> I write an awesome blog using Jupyter notebooks converted to HTML, and I want the images to be cached. Having one html file with all of the images base64 encoded inside it is nice when sharing with a coworker, but for a website, not so much. I need an HTML exporter, and I want it to extract the figures!

3.3.1 Some theory

Before we get into actually extracting the figures, it will be helpful to give a high-level overview of the process of converting a notebook to a another format:

1. Retrieve the notebook and it's accompanying resources (you are responsible for this).
2. Feed the notebook into the `Exporter`, which:
    1. Sequentially feeds the notebook into an array of `Preprocessors`. Preprocessors only act on the **structure** of the notebook, and have unrestricted access to it.
    2. Feeds the notebook into the Jinja templating engine, which converts it to a particular format depending on which template is selected.
3. The exporter returns the converted notebook and other relevant resources as a tuple.
4. You write the data to the disk using the built-in `FilesWriter` (which writes the notebook and any extracted files to disk), or elsewhere using a custom `Writer`.

3.3.2 Using different preprocessors

To extract the figures when using the HTML exporter, we will want to change which `Preprocessors` we are using. There are several preprocessors that come with nbconvert, including one called the `ExtractOutputPreprocessor`.

The `ExtractOutputPreprocessor` is responsible for crawling the notebook, finding all of the figures, and putting them into the resources directory, as well as choosing the key (i.e. `filename_xx_y.extension`) that can replace the figure inside the template. To enable the `ExtractOutputPreprocessor`, we must add it to the exporter's list of preprocessors:

[9]:
```
# create a configuration object that changes the preprocessors
from traitlets.config import Config
c = Config()
c.HTMLExporter.preprocessors = ['nbconvert.preprocessors.ExtractOutputPreprocessor']

# create the new exporter using the custom config
html_exporter_with_figs = HTMLExporter(config=c)
html_exporter_with_figs.preprocessors
```

[9]:
```
['nbconvert.preprocessors.ExtractOutputPreprocessor']
```

We can compare the result of converting the notebook using the original HTML exporter and our new customized one:

[10]:
```
(_, resources)          = html_exporter.from_notebook_node(jake_notebook)
(_, resources_with_fig) = html_exporter_with_figs.from_notebook_node(jake_notebook)

print("resources without figures:")
print(sorted(resources.keys()))

print("\nresources with extracted figures (notice that there's one more field called
↪'outputs'):")
print(sorted(resources_with_fig.keys()))

print("\nthe actual figures are:")
print(sorted(resources_with_fig['outputs'].keys()))
```

```
resources without figures:
['deprecated', 'global_content_filter', 'include_css', 'include_js', 'include_url',
↪'inlining', 'jquery_url', 'metadata', 'output_extension', 'raw_mimetypes', 'require_
↪js_url', 'theme']

resources with extracted figures (notice that there's one more field called 'outputs
↪'):
['deprecated', 'global_content_filter', 'include_css', 'include_js', 'include_url',
↪'inlining', 'jquery_url', 'metadata', 'output_extension', 'outputs', 'raw_mimetypes
↪', 'require_js_url', 'theme']

the actual figures are:
['output_13_1.png', 'output_16_0.png', 'output_18_1.png', 'output_3_0.png', 'output_5_
↪0.png']
```

3.4 Custom Preprocessors

There are an endless number of transformations that you may want to apply to a notebook. In particularly complicated cases, you may want to actually create your own `Preprocessor`. Above, when we customized the list of preprocessors accepted by the `HTMLExporter`, we passed in a string – this can be any valid module name. So, if you create your own preprocessor, you can include it in that same list and it will be used by the exporter.

To create your own preprocessor, you will need to subclass from `nbconvert.preprocessors.Preprocessor` and overwrite either the `preprocess` and/or `preprocess_cell` methods.

3.5 Example

The following demonstration adds the ability to exclude a cell by index.

Note: injecting cells is similar, and won't be covered here. If you want to inject static content at the beginning/end of a notebook, use a custom template.

[11]:
```
from traitlets import Integer
from nbconvert.preprocessors import Preprocessor

class PelicanSubCell(Preprocessor):
    """A Pelican specific preprocessor to remove some of the cells of a notebook"""

    # I could also read the cells from nb.metadata.pelican if someone wrote a JS␣
↪extension,
    # but for now I'll stay with configurable value.
    start = Integer(0,  help="first cell of notebook to be converted").
↪tag(config=True)
    end   = Integer(-1, help="last cell of notebook to be converted").tag(config=True)

    def preprocess(self, nb, resources):
        self.log.info("I'll keep only cells from %d to %d", self.start, self.end)
        nb.cells = nb.cells[self.start:self.end]
        return nb, resources
```

Here a Pelican exporter is created that takes `PelicanSubCell` preprocessors and a `config` object as parameters. This may seem redundant, but with the configuration system you can register an inactive preprocessor on all of the exporters and activate it from config files or the command line.

[12]:
```
# Create a new config object that configures both the new preprocessor, as well as␣
↪the exporter
c =  Config()
c.PelicanSubCell.start = 4
c.PelicanSubCell.end = 6
c.RSTExporter.preprocessors = [PelicanSubCell]

# Create our new, customized exporter that uses our custom preprocessor
pelican = RSTExporter(config=c)

# Process the notebook
print(pelican.from_notebook_node(jake_notebook)[0])
```

```
Sometimes when showing schematic plots, this is the type of figure I
want to display. But drawing it by hand is a pain: I'd rather just use
matplotlib. The problem is, matplotlib is a bit too precise. Attempting
```

(continues on next page)

(continued from previous page)

```
to duplicate this figure in matplotlib leads to something like this:

.. code:: ipython3

    Image('http://jakevdp.github.com/figures/mpl_version.png')



.. image:: output_5_0.png
```

3.6 Programmatically creating templates

[13]:
```
from jinja2 import DictLoader

dl = DictLoader({'footer':
"""
{%- extends 'lab/index.html.j2' -%}

{% block footer %}
FOOOOOOOOTEEEEER
{% endblock footer %}
"""})


exportHTML = HTMLExporter(extra_loaders=[dl], template_file='footer')
(body, resources) = exportHTML.from_notebook_node(jake_notebook)
for l in body.split('\n')[-4:]:
    print(l)
```

```
</body>

FOOOOOOOOTEEEEER
```

3.7 Real World Uses

@jakevdp uses Pelican and Jupyter Notebook to blog. Pelican will use nbconvert programmatically to generate blog post. Have a look a Pythonic Preambulations for Jake's blog post.

@damianavila wrote the Nikola Plugin to write blog post as Notebooks and is developing a js-extension to publish notebooks via one click from the web app.

As @Mbussonn requested... easieeeeer! Deploy your Nikola site with just a click in the IPython notebook! http://t.co/860sJunZvj cc @ralsina

— Damián Avila (@damian_avila) August 21, 2013

CHAPTER

FOUR

LATEX CITATIONS

`nbconvert` now has support for LaTeX citations. With this capability you can:

- Manage citations using BibTeX.
- Cite those citations in Markdown cells using HTML data attributes.
- Have `nbconvert` generate proper LaTeX citations and run BibTeX.

For an example of how this works, please see the citations example in the nbconvert-examples repository.

CHAPTER

FIVE

REMOVING CELLS, INPUTS, OR OUTPUTS

When converting Notebooks into other formats, it is possible to remove parts of a cell, or entire cells, using preprocessors. The notebook will remain unchanged, but the outputs will have certain pieces removed. Here are two primary ways to accomplish this.

5.1 Removing pieces of cells using cell tags

The most straightforward way to control which pieces of cells are removed is to use **cell tags**. These are single-string snippets of metadata that are stored in each cells "tag" field. The *TagRemovePreprocessor* can be used to remove inputs, outputs, or entire cells.

For example, here is a configuration that uses a different tag for removing each part of a cell with the HTMLExporter. In this case, we demonstrate using the nbconvert Python API.

```
from traitlets.config import Config
import nbformat as nbf
from nbconvert.exporters import HTMLExporter

c = Config()

# Configure our tag removal
c.TagRemovePreprocessor.remove_cell_tags = ("remove_cell",)
c.TagRemovePreprocessor.remove_all_outputs_tags = ('remove_output',)
c.TagRemovePreprocessor.remove_input_tags = ('remove_input',)

# Configure and run out exporter
c.HTMLExporter.preprocessors = ["TagRemovePreprocessor"]
HTMLExporter(config=c).from_filename("path/to/mynotebook.ipynb")
```

5.2 Removing cells using Regular Expressions on cell content

Sometimes you'd rather remove cells based on their _content_ rather than their tags. In this case, you can use the *RegexRemovePreprocessor*.

You initalize this preprocessor with a single `patterns` configuration, which is a list of strings. For each cell, this preprocessor checks whether the cell contents match any of the strings provided in `patterns`. If the contents match any of the patterns, the cell is removed from the notebook.

For example, execute the following command to convert a notebook to html and remove cells containing only whitespace:

```
jupyter nbconvert --RegexRemovePreprocessor.patterns="['\s*\Z']" mynotebook.ipynb
```

The command line argument sets the list of patterns to `'\s*\Z'` which matches an arbitrary number of whitespace characters followed by the end of the string.

See https://regex101.com/ for an interactive guide to regular expressions (make sure to select the python flavor). See https://docs.python.org/library/re.html for the official regular expression documentation in python.

CHAPTER

SIX

EXECUTING NOTEBOOKS

Jupyter notebooks are often saved with output cells that have been cleared. nbconvert provides a convenient way to execute the input cells of an .ipynb notebook file and save the results, both input and output cells, as a .ipynb file.

In this section we show how to execute a `.ipynb` notebook document saving the result in notebook format. If you need to export notebooks to other formats, such as reStructured Text or Markdown (optionally executing them) see section *Using nbconvert as a library*.

Executing notebooks can be very helpful, for example, to run all notebooks in Python library in one step, or as a way to automate the data analysis in projects involving more than one notebook.

6.1 Executing notebooks from the command line

The same functionality of executing notebooks is exposed through a *command line interface* or a Python API interface. As an example, a notebook can be executed from the command line with:

```
jupyter nbconvert --to notebook --execute mynotebook.ipynb
```

6.2 Executing notebooks using the Python API interface

This section will illustrate the Python API interface.

6.2.1 Example

Let's start with a complete quick example, leaving detailed explanations to the following sections.

Import: First we import nbconvert and the *`ExecutePreprocessor`* class:

```
import nbformat
from nbconvert.preprocessors import ExecutePreprocessor
```

Load: Assuming that `notebook_filename` contains the path of a notebook, we can load it with:

```
with open(notebook_filename) as f:
    nb = nbformat.read(f, as_version=4)
```

Configure: Next, we configure the notebook execution mode:

```
ep = ExecutePreprocessor(timeout=600, kernel_name='python3')
```

We specified two (optional) arguments `timeout` and `kernel_name`, which define respectively the cell execution timeout and the execution kernel.

> The option to specify **kernel_name** is new in nbconvert 4.2. When not specified or when using nbconvert <4.2, the default Python kernel is chosen.

Execute/Run (preprocess): To actually run the notebook we call the method *preprocess()*:

```
ep.preprocess(nb, {'metadata': {'path': 'notebooks/'}})
```

Hopefully, we will not get any errors during the notebook execution (see the last section for error handling). Note that `path` specifies in which folder to execute the notebook.

Save: Finally, save the resulting notebook with:

```
with open('executed_notebook.ipynb', 'w', encoding='utf-8') as f:
    nbformat.write(nb, f)
```

That's all. Your executed notebook will be saved in the current folder in the file `executed_notebook.ipynb`.

6.3 Execution arguments (traitlets)

The arguments passed to *ExecutePreprocessor* are configuration options called traitlets. There are many cool things about traitlets. For example, they enforce the input type, and they can be accessed/modified as class attributes. Moreover, each traitlet is automatically exposed as command-line options. For example, we can pass the timeout from the command-line like this:

```
jupyter nbconvert --ExecutePreprocessor.timeout=600 --to notebook --execute␣
↪mynotebook.ipynb
```

Let's now discuss in more detail the two traitlets we used.

The `timeout` traitlet defines the maximum time (in seconds) each notebook cell is allowed to run, if the execution takes longer an exception will be raised. The default is 30 s, so in cases of long-running cells you may want to specify an higher value. The `timeout` option can also be set to `None` or `-1` to remove any restriction on execution time.

The second traitlet, `kernel_name`, allows specifying the name of the kernel to be used for the execution. By default, the kernel name is obtained from the notebook metadata. The traitlet `kernel_name` allows specifying a user-defined kernel, overriding the value in the notebook metadata. A common use case is that of a Python 2/3 library which includes documentation/testing notebooks. These notebooks will specify either a python2 or python3 kernel in their metadata (depending on the kernel used the last time the notebook was saved). In reality, these notebooks will work on both Python 2 and Python 3, and, for testing, it is important to be able to execute them programmatically on both versions. Here the traitlet `kernel_name` helps simplify and maintain consistency: we can just run a notebook twice, specifying first "python2" and then "python3" as the kernel name.

6.4 Handling errors and exceptions

In the previous sections we saw how to save an executed notebook, assuming there are no execution errors. But, what if there are errors?

6.4.1 Execution until first error

An error during the notebook execution, by default, will stop the execution and raise a *CellExecutionError*. Conveniently, the source cell causing the error and the original error name and message are also printed. After an error, we can still save the notebook as before:

```
with open('executed_notebook.ipynb', mode='w', encoding='utf-8') as f:
    nbformat.write(nb, f)
```

The saved notebook contains the output up until the failing cell, and includes a full stack-trace and error (which can help debugging).

6.4.2 Handling errors

A useful pattern to execute notebooks while handling errors is the following:

```
from nbconvert.preprocessors import CellExecutionError

try:
    out = ep.preprocess(nb, {'metadata': {'path': run_path}})
except CellExecutionError:
    out = None
    msg = 'Error executing the notebook "%s".\n\n' % notebook_filename
    msg += 'See notebook "%s" for the traceback.' % notebook_filename_out
    print(msg)
    raise
finally:
    with open(notebook_filename_out, mode='w', encoding='utf-8') as f:
        nbformat.write(nb, f)
```

This will save the executed notebook regardless of execution errors. In case of errors, however, an additional message is printed and the *CellExecutionError* is raised. The message directs the user to the saved notebook for further inspection.

6.4.3 Execute and save all errors

As a last scenario, it is sometimes useful to execute notebooks which raise exceptions, for example to show an error condition. In this case, instead of stopping the execution on the first error, we can keep executing the notebook using the traitlet `allow_errors` (default is False). With `allow_errors=True`, the notebook is executed until the end, regardless of any error encountered during the execution. The output notebook, will contain the stack-traces and error messages for **all** the cells raising exceptions.

6.5 Widget state

If your notebook contains any Jupyter Widgets, the state of all the widgets can be stored in the notebook's metadata. This allows rendering of the live widgets on for instance nbviewer, or when converting to html.

We can tell nbconvert to not store the state using the `store_widget_state` argument:

```
jupyter nbconvert --ExecutePreprocessor.store_widget_state=False --to notebook --
↪execute mynotebook.ipynb
```

This widget rendering is not performed against a browser during execution, so only widget default states or states manipulated via user code will be calculated during execution. `%%javascript` cells will execute upon notebook rendering, enabling complex interactions to function as expected when viewed by a UI.

If you can't view widget results after execution, you may need to select *File → Trust Notebook* in the menu.

CHAPTER

SEVEN

CONFIGURATION OPTIONS

Configuration options may be set in a file, `~/.jupyter/jupyter_nbconvert_config.py`, or at the command line when starting nbconvert, i.e. `jupyter nbconvert --Application.log_level=10`.

The most specific setting will always be used. For example, the LatexExporter and the HTMLExporter both inherit from TemplateExporter. With the following config

```
c.TemplateExporter.exclude_input_prompt = False # The default
c.PDFExporter.exclude_input_prompt = True
```

input prompts will not appear when converting to PDF, but they will appear when exporting to HTML.

7.1 CLI Flags and Aliases

When using Nbconvert from the command line, a number of aliases and flags are defined as shortcuts to configuration options for convience.

The following flags are defined:

debug set log level to logging.DEBUG (maximize logging output)

Long Form: {'Application': {'log_level': 10}}

generate-config generate default config file

Long Form: {'JupyterApp': {'generate_config': True}}

y Answer yes to any questions instead of prompting.

Long Form: {'JupyterApp': {'answer_yes': True}}

execute Execute the notebook prior to export.

Long Form: {'ExecutePreprocessor': {'enabled': True}}

allow-errors Continue notebook execution even if one of the cells throws an error and include the error message in the cell output (the default behaviour is to abort conversion). This flag is only relevant if '–execute' was specified, too.

Long Form: {'ExecutePreprocessor': {'allow_errors': True}}

stdin read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'

Long Form: {'NbConvertApp': {'from_stdin': True}}

stdout Write notebook output to stdout instead of files.

Long Form: {'NbConvertApp': {'writer_class': 'StdoutWriter'}}

inplace Run nbconvert in place, overwriting the existing notebook (only relevant when converting to notebook format)

Long Form: {'NbConvertApp': {'use_output_suffix': False, 'export_format': 'notebook'}, 'FilesWriter': {'build_directory': ''}}

clear-output Clear output of current file and save in place, overwriting the existing notebook.

Long Form: {'NbConvertApp': {'use_output_suffix': False, 'export_format': 'notebook'}, 'FilesWriter': {'build_directory': ''}, 'ClearOutputPreprocessor': {'enabled': True}}

no-prompt Exclude input and output prompts from converted document.

Long Form: {'TemplateExporter': {'exclude_input_prompt': True, 'exclude_output_prompt': True}}

no-input Exclude input cells and output prompts from converted document. This mode is ideal for generating code-free reports.

Long Form: {'TemplateExporter': {'exclude_output_prompt': True, 'exclude_input': True}}

allow-chromium-download Whether to allow downloading chromium if no suitable version is found on the system.

Long Form: {'WebPDFExporter': {'allow_chromium_download': True}}

The folowing aliases are defined:

log-level (Application.log_level)

config (JupyterApp.config_file)

to (NbConvertApp.export_format)

template (TemplateExporter.template_name)

template-file (TemplateExporter.template_file)

writer (NbConvertApp.writer_class)

post (NbConvertApp.postprocessor_class)

output (NbConvertApp.output_base)

output-dir (FilesWriter.build_directory)

reveal-prefix (SlidesExporter.reveal_url_prefix)

nbformat (NotebookExporter.nbformat_version)

7.2 App Options

Application.log_datefmt : Unicode Default: `'%Y-%m-%d %H:%M:%S'`

The date format used by logging formatters for %(asctime)s

Application.log_format : Unicode Default: `'[%(name)s]%(highlevel)s %(message)s'`

The Logging format template

Application.log_level : any of 0``|``10``|``20``|``30``|``40``|``50``|'DEBUG'|'INFO'|'WARN'|'ERROR'|' Default: `30`

Set the log level by value or name.

Application.show_config : Bool Default: `False`

Instead of starting the Application, dump configuration to stdout

Application.show_config_json : Bool Default: `False`

Instead of starting the Application, dump configuration to stdout (as JSON)

JupyterApp.answer_yes : Bool Default: `False`

Answer yes to any prompts.

JupyterApp.config_file : Unicode Default: `''`

Full path of a config file.

JupyterApp.config_file_name : Unicode Default: `''`

Specify a config file to load.

JupyterApp.generate_config : Bool Default: `False`

Generate default config file.

JupyterApp.log_datefmt : Unicode Default: `'%Y-%m-%d %H:%M:%S'`

The date format used by logging formatters for %(asctime)s

JupyterApp.log_format : Unicode Default: `'[%(name)s]%(highlevel)s %(message)s'`

The Logging format template

JupyterApp.log_level : any of `0``|``10``|``20``|``30``|``40``|``50``|'DEBUG'|'INFO'|'WARN'|'ERROR'|'`
Default: `30`

Set the log level by value or name.

JupyterApp.show_config : Bool Default: `False`

Instead of starting the Application, dump configuration to stdout

JupyterApp.show_config_json : Bool Default: `False`

Instead of starting the Application, dump configuration to stdout (as JSON)

NbConvertApp.answer_yes : Bool Default: `False`

Answer yes to any prompts.

NbConvertApp.config_file : Unicode Default: `''`

Full path of a config file.

NbConvertApp.config_file_name : Unicode Default: `''`

Specify a config file to load.

NbConvertApp.export_format : Unicode Default: `''`

The export format to be used, either one of the built-in formats ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf'] or a dotted object name that represents the import path for an *Exporter* class

NbConvertApp.from_stdin : Bool Default: `False`

read a single notebook from stdin.

NbConvertApp.generate_config : Bool Default: `False`

Generate default config file.

NbConvertApp.html_manager_semver_range : Unicode Default: `'*'`

Semver range for Jupyter widgets HTML manager

NbConvertApp.jupyter_widgets_base_url : Unicode Default: `'https://unpkg.com/'`

URL base for Jupyter widgets

NbConvertApp.log_datefmt : Unicode Default: `'%Y-%m-%d %H:%M:%S'`

The date format used by logging formatters for %(asctime)s

NbConvertApp.log_format : Unicode Default: `'[%(name)s]%(highlevel)s %(message)s'`

The Logging format template

NbConvertApp.log_level : any of 0``|``10``|``20``|``30``|``40``|``50``|'DEBUG'|'INFO'|'WARN'|'ERROR
Default: `30`

Set the log level by value or name.

NbConvertApp.notebooks : List Default: `[]`

List of notebooks to convert. Wildcards are supported. Filenames passed positionally will be added to the list.

NbConvertApp.output_base : Unicode Default: `''`

overwrite base name use for output files. can only be used when converting one notebook at a time.

NbConvertApp.output_files_dir : Unicode Default: `'{notebook_name}_files'`

Directory to copy extra files (figures) to. '{notebook_name}' in the string will be converted to notebook basename.

NbConvertApp.postprocessor_class : DottedOrNone Default: `''`

PostProcessor class used to write the results of the conversion

NbConvertApp.show_config : Bool Default: `False`

Instead of starting the Application, dump configuration to stdout

NbConvertApp.show_config_json : Bool Default: `False`

Instead of starting the Application, dump configuration to stdout (as JSON)

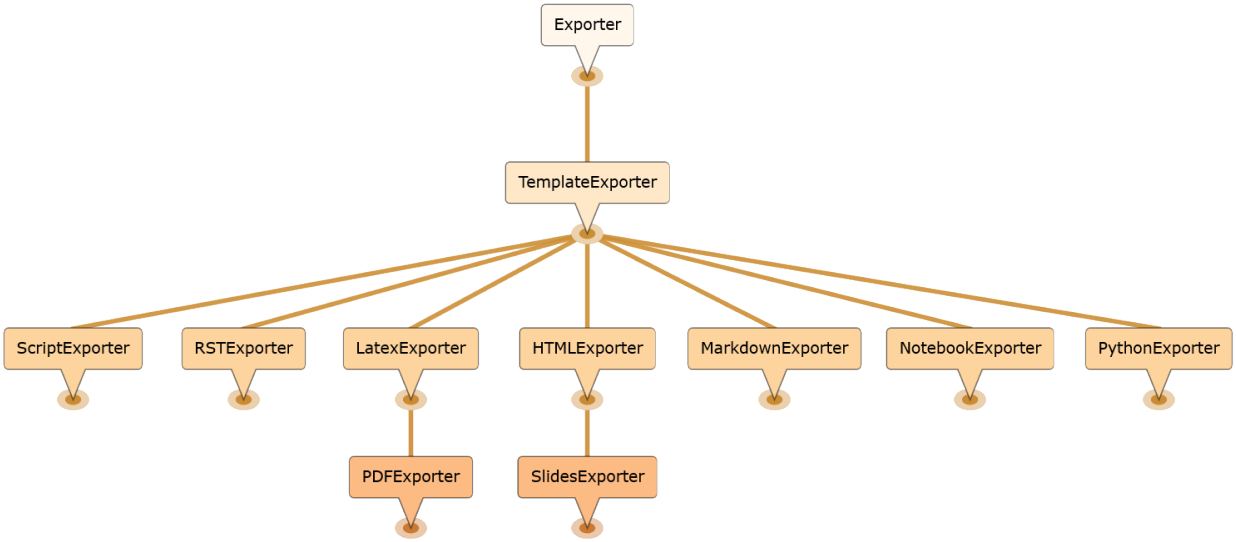
NbConvertApp.use_output_suffix : Bool Default: `True`

Whether to apply a suffix prior to the extension (only relevant when converting to notebook format). The suffix is determined by the exporter, and is usually '.nbconvert'.

NbConvertApp.writer_class : DottedObjectName Default: `'FilesWriter'`

Writer class used to write the results of the conversion

7.3 Exporter Options



Exporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

Exporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

Exporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

Exporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

TemplateExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

TemplateExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

TemplateExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

TemplateExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

TemplateExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

TemplateExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

TemplateExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

TemplateExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

TemplateExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

TemplateExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

TemplateExporter.extra_template_basedirs : List Default: `[]`

No description

TemplateExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

TemplateExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

TemplateExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

TemplateExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

TemplateExporter.template_extension : Unicode Default: `''`

No description

TemplateExporter.template_file : Unicode Default: `None`

Name of the template file to use

TemplateExporter.template_name : Unicode Default: `''`

Name of the template to use

TemplateExporter.template_paths : List Default: `['.']`

No description

ASCIIDocExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

ASCIIDocExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

ASCIIDocExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

ASCIIDocExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

ASCIIDocExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

ASCIIDocExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

ASCIIDocExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

ASCIIDocExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

ASCIIDocExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

ASCIIDocExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

ASCIIDocExporter.extra_template_basedirs : List Default: `[]`

No description

ASCIIDocExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

ASCIIDocExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

ASCIIDocExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

ASCIIDocExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

ASCIIDocExporter.template_extension : Unicode Default: `''`

No description

ASCIIDocExporter.template_file : Unicode Default: `None`

Name of the template file to use

ASCIIDocExporter.template_name : Unicode Default: `''`

Name of the template to use

ASCIIDocExporter.template_paths : List Default: `['.']`

No description

HTMLExporter.anchor_link_text : Unicode Default: `'¶'`

The text used as the text for anchor links.

HTMLExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

HTMLExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

HTMLExporter.exclude_anchor_links : Bool Default: `False`

If anchor links should be included or not.

HTMLExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

HTMLExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

HTMLExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

HTMLExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

HTMLExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

HTMLExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

HTMLExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

HTMLExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

HTMLExporter.extra_template_basedirs : List Default: `[]`

No description

HTMLExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

HTMLExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

HTMLExporter.jquery_url : Unicode Default: `'https://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.m...`

URL to load jQuery from.

Defaults to loading from cdnjs.

HTMLExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

HTMLExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

HTMLExporter.require_js_url : Unicode Default: `'https://cdnjs.cloudflare.com/ajax/libs/require.js/2.1.10/req...`

URL to load require.js from.

Defaults to loading from cdnjs.

HTMLExporter.template_extension : Unicode Default: `''`

No description

HTMLExporter.template_file : Unicode Default: `None`

Name of the template file to use

HTMLExporter.template_name : Unicode Default: `''`

Name of the template to use

HTMLExporter.template_paths : List Default: `['.']`

No description

HTMLExporter.theme : Unicode Default: `'light'`

Template specific theme(e.g. the JupyterLab CSS theme for the lab template)

LatexExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

LatexExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

LatexExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

LatexExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

LatexExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

LatexExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

LatexExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

LatexExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

LatexExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

LatexExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

LatexExporter.extra_template_basedirs : List Default: `[]`

No description

LatexExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

LatexExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

LatexExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

LatexExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

LatexExporter.template_extension : Unicode Default: `''`

No description

LatexExporter.template_file : Unicode Default: `None`

Name of the template file to use

LatexExporter.template_name : Unicode Default: `''`

Name of the template to use

LatexExporter.template_paths : List Default: `['.']`

No description

MarkdownExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

MarkdownExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

MarkdownExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

MarkdownExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

MarkdownExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

MarkdownExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

MarkdownExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

MarkdownExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

MarkdownExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

MarkdownExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

MarkdownExporter.extra_template_basedirs : List Default: `[]`

No description

MarkdownExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

MarkdownExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

MarkdownExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

MarkdownExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

MarkdownExporter.template_extension : Unicode Default: `''`

No description

MarkdownExporter.template_file : Unicode Default: `None`

Name of the template file to use

MarkdownExporter.template_name : Unicode Default: `''`

Name of the template to use

MarkdownExporter.template_paths : List Default: `['.']`

No description

NotebookExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

NotebookExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

NotebookExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

NotebookExporter.nbformat_version : any of 1``|``2``|``3``|``4 Default: `4`

The nbformat version to write. Use this to downgrade notebooks.

NotebookExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

PDFExporter.bib_command : List Default: `['bibtex', '{filename}']`

Shell command used to run bibtex.

PDFExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

PDFExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

PDFExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

PDFExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

PDFExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

PDFExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

PDFExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

PDFExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

PDFExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

PDFExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

PDFExporter.extra_template_basedirs : List Default: `[]`

No description

PDFExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

PDFExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

PDFExporter.latex_command : List Default: `['xelatex', '{filename}', '-quiet']`

Shell command used to compile latex.

PDFExporter.latex_count : Int Default: `3`

How many times latex will be called.

PDFExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

PDFExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

PDFExporter.template_extension : Unicode Default: `''`

No description

PDFExporter.template_file : Unicode Default: `None`

Name of the template file to use

PDFExporter.template_name : Unicode Default: `''`

Name of the template to use

PDFExporter.template_paths : List Default: `['.']`

No description

PDFExporter.verbose : Bool Default: `False`

Whether to display the output of latex commands.

PythonExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

PythonExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

PythonExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

PythonExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

PythonExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

PythonExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

PythonExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

PythonExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

PythonExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

PythonExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

PythonExporter.extra_template_basedirs : List Default: `[]`

No description

PythonExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

PythonExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

PythonExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

PythonExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

PythonExporter.template_extension : Unicode Default: `''`

No description

PythonExporter.template_file : Unicode Default: `None`

Name of the template file to use

PythonExporter.template_name : Unicode Default: `''`

Name of the template to use

PythonExporter.template_paths : List Default: `['.']`

No description

RSTExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

RSTExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

RSTExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

RSTExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

RSTExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

RSTExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

RSTExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

RSTExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

RSTExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

RSTExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

RSTExporter.extra_template_basedirs : List Default: `[]`

No description

RSTExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

RSTExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

RSTExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

RSTExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

RSTExporter.template_extension : Unicode Default: `''`

No description

RSTExporter.template_file : Unicode Default: `None`

Name of the template file to use

RSTExporter.template_name : Unicode Default: `''`

Name of the template to use

RSTExporter.template_paths : List Default: `['.']`

No description

ScriptExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

ScriptExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

ScriptExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

ScriptExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

ScriptExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

ScriptExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

ScriptExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

ScriptExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

ScriptExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

ScriptExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

ScriptExporter.extra_template_basedirs : List Default: `[]`

No description

ScriptExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

ScriptExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

ScriptExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

ScriptExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

ScriptExporter.template_extension : Unicode Default: `''`

No description

ScriptExporter.template_file : Unicode Default: `None`

Name of the template file to use

ScriptExporter.template_name : Unicode Default: `''`

Name of the template to use

ScriptExporter.template_paths : List Default: `['.']`

No description

SlidesExporter.anchor_link_text : Unicode Default: `'¶'`

The text used as the text for anchor links.

SlidesExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

SlidesExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

SlidesExporter.exclude_anchor_links : Bool Default: `False`

If anchor links should be included or not.

SlidesExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

SlidesExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

SlidesExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

SlidesExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

SlidesExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

SlidesExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

SlidesExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

SlidesExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

SlidesExporter.extra_template_basedirs : List Default: `[]`

No description

SlidesExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

SlidesExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

SlidesExporter.font_awesome_url : Unicode Default: `'https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/cs...`

URL to load font awesome from.

Defaults to loading from cdnjs.

SlidesExporter.jquery_url : Unicode Default: `'https://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.m...`

URL to load jQuery from.

Defaults to loading from cdnjs.

SlidesExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

SlidesExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

SlidesExporter.require_js_url : Unicode Default: `'https://cdnjs.cloudflare.com/ajax/libs/require.js/2.1.10/req...`

URL to load require.js from.

Defaults to loading from cdnjs.

SlidesExporter.reveal_scroll : Bool Default: `False`

If True, enable scrolling within each slide

SlidesExporter.reveal_theme : Unicode Default: `'simple'`

Name of the reveal.js theme to use.

We look for a file with this name under `reveal_url_prefix`/css/theme/`reveal_theme`.css.

https://github.com/hakimel/reveal.js/tree/master/css/theme has list of themes that ship by default with reveal.js.

SlidesExporter.reveal_transition : Unicode Default: `'slide'`

Name of the reveal.js transition to use.

The list of transitions that ships by default with reveal.js are: none, fade, slide, convex, concave and zoom.

SlidesExporter.reveal_url_prefix : Unicode Default: `''`

The URL prefix for reveal.js (version 3.x). This defaults to the reveal CDN, but can be any url pointing to a copy of reveal.js.

For speaker notes to work, this must be a relative path to a local copy of reveal.js: e.g., "reveal.js".

If a relative path is given, it must be a subdirectory of the current directory (from which the server is run).

See the usage documentation (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-slideshow) for more details.

SlidesExporter.template_extension : Unicode Default: `''`

No description

SlidesExporter.template_file : Unicode Default: `None`

Name of the template file to use

SlidesExporter.template_name : Unicode Default: `''`

Name of the template to use

SlidesExporter.template_paths : List Default: `['.']`

No description

SlidesExporter.theme : Unicode Default: `'light'`

Template specific theme(e.g. the JupyterLab CSS theme for the lab template)

WebPDFExporter.allow_chromium_download : Bool Default: `False`

Whether to allow downloading Chromium if no suitable version is found on the system.

WebPDFExporter.anchor_link_text : Unicode Default: `'¶'`

The text used as the text for anchor links.

WebPDFExporter.default_preprocessors : List Default: `['nbconvert.preprocessors.TagRemovePreprocessor', 'nbconvert....`

List of preprocessors available by default, by name, namespace, instance, or type.

WebPDFExporter.enabled : Bool Default: `True`

Disable this exporter (and any exporters inherited from it).

WebPDFExporter.exclude_anchor_links : Bool Default: `False`

If anchor links should be included or not.

WebPDFExporter.exclude_code_cell : Bool Default: `False`

This allows you to exclude code cells from all templates if set to True.

WebPDFExporter.exclude_input : Bool Default: `False`

This allows you to exclude code cell inputs from all templates if set to True.

WebPDFExporter.exclude_input_prompt : Bool Default: `False`

This allows you to exclude input prompts from all templates if set to True.

WebPDFExporter.exclude_markdown : Bool Default: `False`

This allows you to exclude markdown cells from all templates if set to True.

WebPDFExporter.exclude_output : Bool Default: `False`

This allows you to exclude code cell outputs from all templates if set to True.

WebPDFExporter.exclude_output_prompt : Bool Default: `False`

This allows you to exclude output prompts from all templates if set to True.

WebPDFExporter.exclude_raw : Bool Default: `False`

This allows you to exclude raw cells from all templates if set to True.

WebPDFExporter.exclude_unknown : Bool Default: `False`

This allows you to exclude unknown cells from all templates if set to True.

WebPDFExporter.extra_template_basedirs : List Default: `[]`

No description

WebPDFExporter.file_extension : FilenameExtension Default: `''`

Extension of the file that should be written to disk

WebPDFExporter.filters : Dict Default: `{}`

Dictionary of filters, by name and namespace, to add to the Jinja environment.

WebPDFExporter.jquery_url : Unicode Default: `'https://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.m...`

URL to load jQuery from.

Defaults to loading from cdnjs.

WebPDFExporter.preprocessors : List Default: `[]`

List of preprocessors, by name or namespace, to enable.

WebPDFExporter.raw_mimetypes : List Default: `[]`

formats of raw cells to be included in this Exporter's output.

WebPDFExporter.require_js_url : Unicode Default: `'https://cdnjs.cloudflare.com/ajax/libs/require.js/2.1.10/req...`

URL to load require.js from.

Defaults to loading from cdnjs.

WebPDFExporter.template_extension : Unicode Default: `''`

No description

WebPDFExporter.template_file : Unicode Default: `None`

Name of the template file to use

WebPDFExporter.template_name : Unicode Default: `''`

Name of the template to use

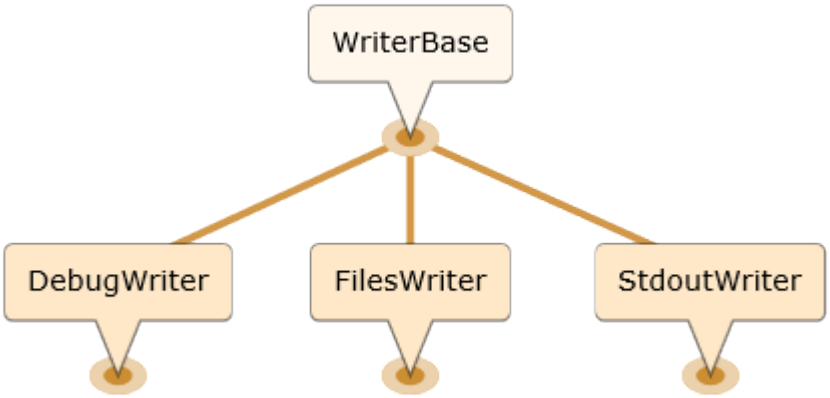
WebPDFExporter.template_paths : List Default: `['.']`

No description

WebPDFExporter.theme : Unicode Default: `'light'`

Template specific theme(e.g. the JupyterLab CSS theme for the lab template)

7.4 Writer Options



WriterBase.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

WriterBase.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

WriterBase.files : List Default: `[]`

List of the files that the notebook references. Files will be included with written output.

DebugWriter.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

DebugWriter.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

DebugWriter.files : List Default: `[]`

List of the files that the notebook references. Files will be included with written output.

FilesWriter.build_directory : Unicode Default: `''`

Directory to write output(s) to. Defaults to output to the directory of each notebook. To recover previous default behaviour (outputting to the current working directory) use . as the flag value.

FilesWriter.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

FilesWriter.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

FilesWriter.files : List Default: `[]`

List of the files that the notebook references. Files will be included with written output.

FilesWriter.relpath : Unicode Default: `''`

When copying files that the notebook depends on, copy them in relation to this path, such that the destination filename will be os.path.relpath(filename, relpath). If FilesWriter is operating on a notebook that already exists elsewhere on disk, then the default will be the directory containing that notebook.

StdoutWriter.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

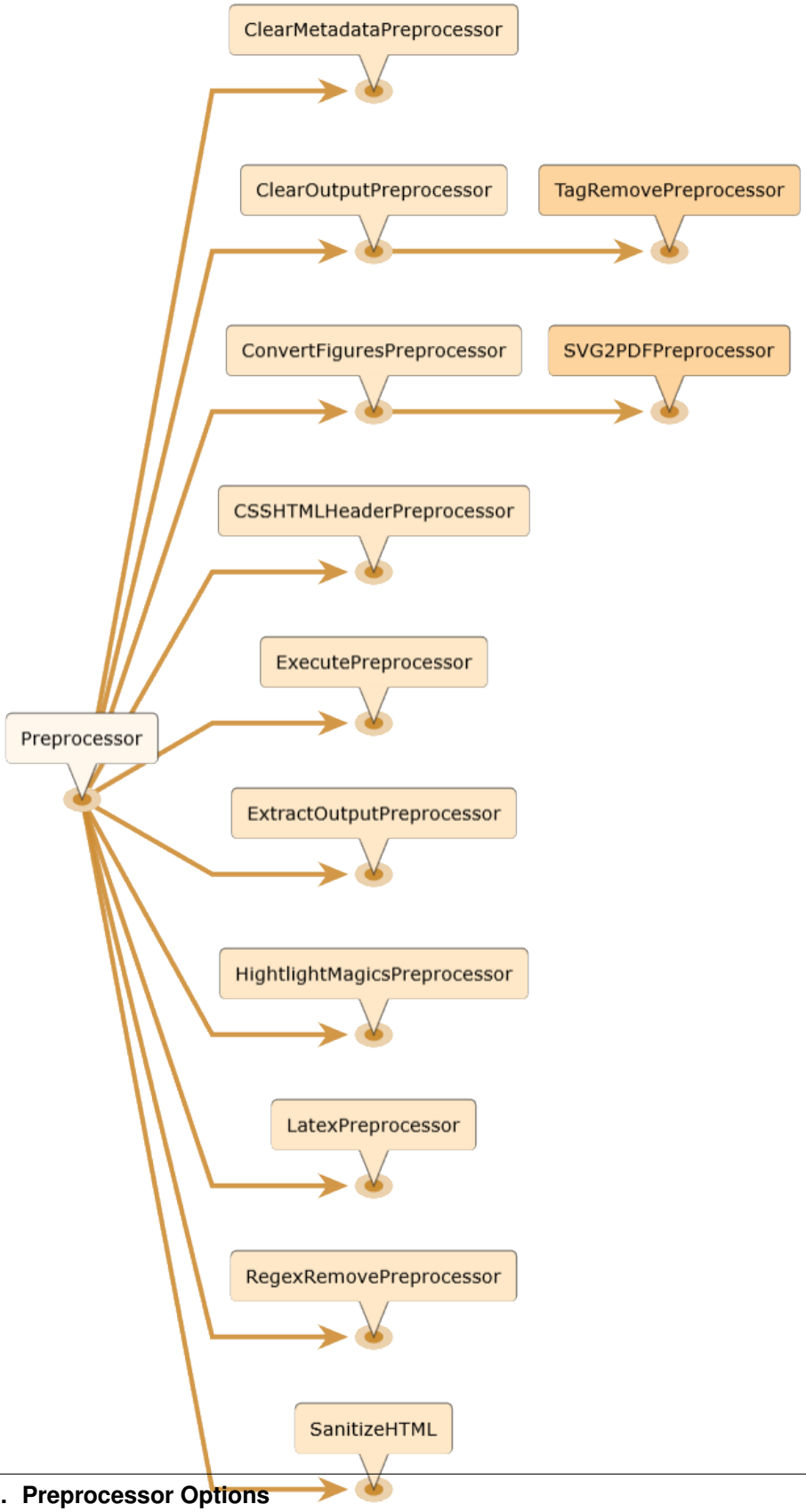
StdoutWriter.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

StdoutWriter.files : List Default: `[]`

List of the files that the notebook references. Files will be included with written output.

7.5 Preprocessor Options



Preprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

Preprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

Preprocessor.enabled : Bool Default: `False`

No description

CSSHTMLHeaderPreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

CSSHTMLHeaderPreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

CSSHTMLHeaderPreprocessor.enabled : Bool Default: `False`

No description

CSSHTMLHeaderPreprocessor.highlight_class : Unicode Default: `'.highlight'`

CSS highlight class identifier

CSSHTMLHeaderPreprocessor.style : Union Default: `<class 'jupyterlab_pygments.style.JupyterStyle'>`

Name of the pygments style to use

ClearMetadataPreprocessor.clear_cell_metadata : Bool Default: `True`

Flag to choose if cell metadata is to be cleared in addition to notebook metadata.

ClearMetadataPreprocessor.clear_notebook_metadata : Bool Default: `True`

Flag to choose if notebook metadata is to be cleared in addition to cell metadata.

ClearMetadataPreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

ClearMetadataPreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

ClearMetadataPreprocessor.enabled : Bool Default: `False`

No description

ClearMetadataPreprocessor.preserve_cell_metadata_mask : Set Default: `set()`

Indicates the key paths to preserve when deleting metadata across both cells and notebook metadata fields. Tuples of keys can be passed to preserved specific nested values

ClearMetadataPreprocessor.preserve_nb_metadata_mask : Set Default: `{('language_info', 'name')}`

Indicates the key paths to preserve when deleting metadata across both cells and notebook metadata fields. Tuples of keys can be passed to preserved specific nested values

ClearOutputPreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

ClearOutputPreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

ClearOutputPreprocessor.enabled : Bool Default: `False`

No description

ClearOutputPreprocessor.remove_metadata_fields : Set Default: `{'collapsed', 'scrolled'}`

No description

ConvertFiguresPreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

ConvertFiguresPreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

ConvertFiguresPreprocessor.enabled : Bool Default: `False`

No description

ConvertFiguresPreprocessor.from_format : Unicode Default: `''`

Format the converter accepts

ConvertFiguresPreprocessor.to_format : Unicode Default: `''`

Format the converter writes

ExecutePreprocessor.allow_errors : Bool Default: `False`

If `False` (default), when a cell raises an error the execution is stopped and a *CellExecutionError* is raised. If `True`, execution errors are ignored and the execution is continued until the end of the notebook. Output from exceptions is included in the cell output in both cases.

ExecutePreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

ExecutePreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

ExecutePreprocessor.enabled : Bool Default: `False`

No description

ExecutePreprocessor.extra_arguments : List Default: `[]`

No description

ExecutePreprocessor.force_raise_errors : Bool Default: `False`

If False (default), errors from executing the notebook can be allowed with a `raises-exception` tag on a single cell, or the `allow_errors` configurable option for all cells. An allowed error will be recorded in notebook output, and execution will continue. If an error occurs when it is not explicitly allowed, a *CellExecutionError* will be raised. If True, *CellExecutionError* will be raised for any error that occurs while executing the notebook. This overrides both the `allow_errors` option and the `raises-exception` cell tag.

ExecutePreprocessor.interrupt_on_timeout : Bool Default: `False`

If execution of a cell times out, interrupt the kernel and continue executing other cells rather than throwing an error and stopping.

ExecutePreprocessor.iopub_timeout : Int Default: `4`

The time to wait (in seconds) for IOPub output. This generally doesn't need to be set, but on some slow networks (such as CI systems) the default timeout might not be long enough to get all messages.

ExecutePreprocessor.ipython_hist_file : Unicode Default: `':memory:'`

Path to file to use for SQLite history database for an IPython kernel.

The specific value `:memory:` (including the colon at both end but not the back ticks), avoids creating a history file. Otherwise, IPython will create a history file for each kernel.

When running kernels simultaneously (e.g. via multiprocessing) saving history a single SQLite file can result in database errors, so using `:memory:` is recommended in non-interactive contexts.

ExecutePreprocessor.kernel_manager_class : Type Default: `'builtins.object'`

The kernel manager class to use.

ExecutePreprocessor.kernel_name : Unicode Default: `''`

Name of kernel to use to execute the cells. If not set, use the kernel_spec embedded in the notebook.

ExecutePreprocessor.raise_on_iopub_timeout : Bool Default: `False`

If `False` (default), then the kernel will continue waiting for iopub messages until it receives a kernel idle message, or until a timeout occurs, at which point the currently executing cell will be skipped. If `True`, then an error will be raised after the first timeout. This option generally does not need to be used, but may be useful in contexts where there is the possibility of executing notebooks with memory-consuming infinite loops.

ExecutePreprocessor.record_timing : Bool Default: `True`

If `True` (default), then the execution timings of each cell will be stored in the metadata of the notebook.

ExecutePreprocessor.shell_timeout_interval : Int Default: `5`

The time to wait (in seconds) for Shell output before retrying. This generally doesn't need to be set, but if one needs to check for dead kernels at a faster rate this can help.

**ExecutePreprocessor.shutdown_kernel : any of `'graceful'`|'immediate'`** Default: `'graceful'`

If `graceful` (default), then the kernel is given time to clean up after executing all cells, e.g., to execute its `atexit` hooks. If `immediate`, then the kernel is signaled to immediately terminate.

ExecutePreprocessor.startup_timeout : Int Default: `60`

The time to wait (in seconds) for the kernel to start. If kernel startup takes longer, a RuntimeError is raised.

ExecutePreprocessor.store_widget_state : Bool Default: `True`

If `True` (default), then the state of the Jupyter widgets created at the kernel will be stored in the metadata of the notebook.

ExecutePreprocessor.timeout : Int Default: `None`

The time to wait (in seconds) for output from executions. If a cell execution takes longer, a TimeoutError is raised.

`None` or `-1` will disable the timeout. If `timeout_func` is set, it overrides `timeout`.

ExecutePreprocessor.timeout_func : Any Default: `None`

A callable which, when given the cell source as input, returns the time to wait (in seconds) for output from cell executions. If a cell execution takes longer, a TimeoutError is raised.

Returning `None` or `-1` will disable the timeout for the cell. Not setting `timeout_func` will cause the preprocessor to default to using the `timeout` trait for all cells. The `timeout_func` trait overrides `timeout` if it is not `None`.

ExtractOutputPreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

ExtractOutputPreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

ExtractOutputPreprocessor.enabled : Bool Default: `False`

No description

ExtractOutputPreprocessor.extract_output_types : Set Default: `{'application/pdf', 'image/jpeg', 'image/png', 'image/svg+xml'}`

No description

ExtractOutputPreprocessor.output_filename_template : Unicode Default: `'{unique_key}_{cell_index}_{index}{ext`

No description

HighlightMagicsPreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

HighlightMagicsPreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

HighlightMagicsPreprocessor.enabled : Bool Default: `False`

No description

HighlightMagicsPreprocessor.languages : Dict Default: `{}`

Syntax highlighting for magic's extension languages. Each item associates a language magic extension such as %%R, with a pygments lexer such as r.

LatexPreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

LatexPreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

LatexPreprocessor.enabled : Bool Default: `False`

No description

LatexPreprocessor.style : Unicode Default: `'default'`

Name of the pygments style to use

RegexRemovePreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

RegexRemovePreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

RegexRemovePreprocessor.enabled : Bool Default: `False`

No description

RegexRemovePreprocessor.patterns : List Default: `[]`

No description

SVG2PDFPreprocessor.command : Unicode Default: `''`

The command to use for converting SVG to PDF

This string is a template, which will be formatted with the keys to_filename and from_filename.

The conversion call must read the SVG from {from_filename}, and write a PDF to {to_filename}.

SVG2PDFPreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

SVG2PDFPreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

SVG2PDFPreprocessor.enabled : Bool Default: `False`

No description

SVG2PDFPreprocessor.from_format : Unicode Default: `''`

Format the converter accepts

SVG2PDFPreprocessor.inkscape : Unicode Default: `''`

The path to Inkscape, if necessary

SVG2PDFPreprocessor.inkscape_version : Unicode Default: `''`

The version of inkscape being used.

This affects how the conversion command is run.

SVG2PDFPreprocessor.to_format : Unicode Default: `''`

Format the converter writes

TagRemovePreprocessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

TagRemovePreprocessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

TagRemovePreprocessor.enabled : Bool Default: `False`

No description

TagRemovePreprocessor.remove_all_outputs_tags : Set Default: `set()`

Tags indicating cells for which the outputs are to be removed,matches tags in `cell.metadata.tags`.

TagRemovePreprocessor.remove_cell_tags : Set Default: `set()`

Tags indicating which cells are to be removed,matches tags in `cell.metadata.tags`.

TagRemovePreprocessor.remove_input_tags : Set Default: `set()`

Tags indicating cells for which input is to be removed,matches tags in `cell.metadata.tags`.

TagRemovePreprocessor.remove_metadata_fields : Set Default: `{'collapsed', 'scrolled'}`

No description

TagRemovePreprocessor.remove_single_output_tags : Set Default: `set()`

Tags indicating which individual outputs are to be removed,matches output *i* tags in `cell.outputs[i].metadata.tags`.

7.6 Postprocessor Options

PostProcessorBase.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

PostProcessorBase.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

ServePostProcessor.browser : Unicode Default: `''`

Specify what browser should be used to open slides. See https://docs.python.org/3/library/webbrowser.html#webbrowser.register to see how keys are mapped to browser executables. If not specified, the default browser will be determined by the `webbrowser` standard library module, which allows setting of the BROWSER environment variable to override it.

ServePostProcessor.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

ServePostProcessor.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

ServePostProcessor.ip : Unicode Default: `'127.0.0.1'`

The IP address to listen on.

ServePostProcessor.open_in_browser : Bool Default: `True`

Should the browser be opened automatically?

ServePostProcessor.port : Int Default: `8000`

port for the server to listen on.

ServePostProcessor.reveal_cdn : Unicode Default: `'https://cdnjs.cloudflare.com/ajax/libs/reveal.js/3.5.0'`

URL for reveal.js CDN.

ServePostProcessor.reveal_prefix : Unicode Default: `'reveal.js'`

URL prefix for reveal.js

7.7 Other Options

NbConvertBase.default_language : Unicode Default: `'ipython'`

Deprecated default highlight language as of 5.0, please use language_info metadata instead

NbConvertBase.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

NotebookClient.allow_errors : Bool Default: `False`

If `False` (default), when a cell raises an error the execution is stopped and a *CellExecutionError* is raised. If `True`, execution errors are ignored and the execution is continued until the end of the notebook. Output from exceptions is included in the cell output in both cases.

NotebookClient.display_data_priority : List Default: `['text/html', 'application/pdf', 'text/latex', 'image/svg+xml...`

An ordered list of preferred output type, the first encountered will usually be used when converting discarding the others.

NotebookClient.extra_arguments : List Default: `[]`

No description

NotebookClient.force_raise_errors : Bool Default: `False`

If False (default), errors from executing the notebook can be allowed with a `raises-exception` tag on a single cell, or the `allow_errors` configurable option for all cells. An allowed error will be recorded in notebook output, and execution will continue. If an error occurs when it is not explicitly allowed, a *CellExecutionError* will be raised. If True, *CellExecutionError* will be raised for any error that occurs while executing the notebook. This overrides both the `allow_errors` option and the `raises-exception` cell tag.

NotebookClient.interrupt_on_timeout : Bool Default: `False`

If execution of a cell times out, interrupt the kernel and continue executing other cells rather than throwing an error and stopping.

NotebookClient.iopub_timeout : Int Default: `4`

The time to wait (in seconds) for IOPub output. This generally doesn't need to be set, but on some slow networks (such as CI systems) the default timeout might not be long enough to get all messages.

NotebookClient.ipython_hist_file : Unicode Default: `':memory:'`

Path to file to use for SQLite history database for an IPython kernel.

The specific value `:memory:` (including the colon at both end but not the back ticks), avoids creating a history file. Otherwise, IPython will create a history file for each kernel.

When running kernels simultaneously (e.g. via multiprocessing) saving history a single SQLite file can result in database errors, so using `:memory:` is recommended in non-interactive contexts.

NotebookClient.kernel_manager_class : Type Default: `'builtins.object'`

The kernel manager class to use.

NotebookClient.kernel_name : Unicode Default: `''`

Name of kernel to use to execute the cells. If not set, use the kernel_spec embedded in the notebook.

NotebookClient.raise_on_iopub_timeout : Bool Default: `False`

If `False` (default), then the kernel will continue waiting for iopub messages until it receives a kernel idle message, or until a timeout occurs, at which point the currently executing cell will be skipped. If `True`, then an error will be raised after the first timeout. This option generally does not need to be used, but may be useful in contexts where there is the possibility of executing notebooks with memory-consuming infinite loops.

NotebookClient.record_timing : Bool Default: `True`

If `True` (default), then the execution timings of each cell will be stored in the metadata of the notebook.

NotebookClient.shell_timeout_interval : Int Default: `5`

The time to wait (in seconds) for Shell output before retrying. This generally doesn't need to be set, but if one needs to check for dead kernels at a faster rate this can help.

NotebookClient.shutdown_kernel : any of `'graceful'`|`'immediate'` Default: `'graceful'`

If `graceful` (default), then the kernel is given time to clean up after executing all cells, e.g., to execute its `atexit` hooks. If `immediate`, then the kernel is signaled to immediately terminate.

NotebookClient.startup_timeout : Int Default: `60`

The time to wait (in seconds) for the kernel to start. If kernel startup takes longer, a RuntimeError is raised.

NotebookClient.store_widget_state : Bool Default: `True`

If `True` (default), then the state of the Jupyter widgets created at the kernel will be stored in the metadata of the notebook.

NotebookClient.timeout : Int Default: `None`

The time to wait (in seconds) for output from executions. If a cell execution takes longer, a TimeoutError is raised.

`None` or `-1` will disable the timeout. If `timeout_func` is set, it overrides `timeout`.

NotebookClient.timeout_func : Any Default: `None`

A callable which, when given the cell source as input, returns the time to wait (in seconds) for output from cell executions. If a cell execution takes longer, a TimeoutError is raised.

Returning `None` or `-1` will disable the timeout for the cell. Not setting `timeout_func` will cause the preprocessor to default to using the `timeout` trait for all cells. The `timeout_func` trait overrides `timeout` if it is not `None`.

CHAPTER

EIGHT

CREATING CUSTOM TEMPLATES FOR NBCONVERT

8.1 Selecting a template

Most exporters in nbconvert are subclasses of *TemplateExporter*, and make use of jinja to render notebooks into the destination format.

Alternative nbconvert templates can be selected by name from the command line with the `--template` option. For example, to use the `reveal` template with the HTML exporter, one can type.

```
jupyter nbconvert <path-to-notebook> --to html --template reveal
```

8.2 Where are nbconvert templates installed?

Nbconvert templates are *directories* containing resources for nbconvert template exporters such as jinja templates and associated assets. They are installed in the **data directory** of nbconvert, namely `<installation prefix>/share/jupyter/nbconvert`. Nbconvert includes several templates already.

For example, three HTML templates are provided in nbconvert core for the HTML exporter:

- `lab` (The default HTML template, which produces the same DOM structure as JupyterLab)
- `classic` (The HTML template styled after the classic notebook)
- `reveal` (For producing slideshows).

Note: Running `jupyter --paths` will show all Jupyter directories and search paths.

For example, on Linux, `jupyter --paths` returns:

```
$ jupyter --paths
config:
    /home/<username>/.jupyter
    /<sys-prefix>/etc/jupyter
    /usr/local/etc/jupyter
    /etc/jupyter
data:
    /home/<username>/.local/share/jupyter
    /<sys-prefix>/share/jupyter
    /usr/local/share/jupyter
    /usr/share/jupyter
runtime:
    /home/<username>/.local/share/jupyter/runtime
```

8.3 The content of nbconvert templates

8.3.1 conf.json

Nbconvert templates all include a `conf.json` file at the root of the directory, which is used to indicate

- the base template that it is inheriting from.
- the mimetypes of the template.
- preprocessors classes to register in the exporter when using that template.

Inspecting the configuration of the reveal template we see that it inherits from the lab template, exports text/html, and enables a preprocessor called "500-reveal".

```
{
  "base_template": "lab",
  "mimetypes": {
    "text/html": true
  },
  "preprocessors": {
    "500-reveal": {
      "type": "nbconvert.exporters.slides._RevealMetadataPreprocessor",
      "enabled": true
    }
  }
}
```

8.3.2 Inheritance

Nbconvert walks up the inheritance structure determined by `conf.json` and produces an agregated configuration, merging the dictionaries of registered preprocessors. The lexical ordering of the preprocessors by name determines the order in which they will be run.

Besides the `conf.json` file, nbconvert templates most typically include jinja templates files, although any other resource from the base template can be overriden in the derived template.

For example, inspecting the content of the `classic` template located in `share/jupyter/nbconvert/templates/classic`, we find the following content:

Inspecting the content of the `classic` template we find the following file structure:

```
share/jupyter/nbconvert/templates/classic
├── static
│   └── styles.css
├── conf.json
├── index.html.j2
└── base.html.j2
```

The `classic` template exporter includes a `index.html.j2` jinja template (which is the main entry point for HTML exporters) as well as CSS and a base template file in `base.html.j2`.

Note: A template inheriting from `classic` would specify `"base_template": "classic"` and could override any of these files. For example, one could make a "classiker" template merely providing an alternative `styles.css` file.

8.3.3 Inheritance in Jinja

In nbconvert, jinja templates can inherrit from any other jinja template available in its current directory or base template directory by name. Jinja templates of other directories can be addressed by their relative path from the Jupyter data directory.

For example, in the reveal template, `index.html.j2` extends `base.html.j2` which is in the same directory, and `base.html.j2` extends `lab/base.html.j2`. This approach allows using content that is available in other templates or may be overriden in the current template.

CHAPTER

NINE

CUSTOMIZING EXPORTERS

New in version 4.2: You can now use the `--to` flag to use custom export formats defined outside nbconvert.

The command-line syntax to run the `nbconvert` script is:

```
jupyter nbconvert --to FORMAT notebook.ipynb
```

This will convert the Jupyter document file `notebook.ipynb` into the output format designated by the `FORMAT` string as explained below.

9.1 Extending the built-in format exporters

A few built-in formats are available by default: `html`, `pdf`, `webpdf`, `script`, `latex`. Each of these has its own *exporter* with many configuration options that can be extended. Having the option to point to a different *exporter* allows authors to create their own fully customized templates or export formats.

A custom *exporter* must be an importable Python object. We recommend that these be distributed as Python libraries.

9.2 Registering a custom exporter as an entry point

Additional exporters may be registered as named entry_points. nbconvert uses the `nbconvert.exporters` entry point to find exporters from any package you may have installed.

If you are writing a Python package that provides custom exporters, you can register the custom exporters in your package's `setup.py`. For example, your package may contain two custom exporters, named "simple" and "detail", and can be registered in your package's `setup.py` as follows:

```
setup(
    ...
    entry_points = {
        'nbconvert.exporters': [
            'simple = mymodule:SimpleExporter',
            'detail = mymodule:DetailExporter',
        ],
    }
)
```

Now people who have installed your Python package containing the two custom exporters can call the entry point name:

```
jupyter nbconvert --to detail mynotebook.ipynb
```

instead of having to specify the full import name of the custom exporter.

9.3 Using a custom exporter without entrypoints

We encourage registering custom exporters as entry points as described in the previous section. Registering a custom exporter with an entry point simplifies using the exporter. If a custom exporter has not been registered with an entry point, the exporter can still be used by providing the fully qualified name of this exporter as the argument of the `--to` flag when running from the command line:

```
$ jupyter nbconvert --to <full.qualified.name of custom exporter> notebook.ipynb
```

For example, assuming a library `tcontrib` has a custom exporter name `TExporter`, you would convert to this custom format using the following:

```
$ jupyter nbconvert --to tcontrib.TExporter notebook.ipynb
```

A library can contain multiple exporters. Creators of custom exporters should make sure that all other flags of the command line behave the same for the custom exporters as for built-in exporters.

CHAPTER

TEN

PARAMETERS CONTROLLED BY AN EXTERNAL EXPORTER

An external exporter can control almost any parameter of the notebook conversion process, from simple parameters such as the output file extension, to more complex ones such as the execution of the notebook or a custom rendering template.

All external exporters can expose custom options using the `traitlets` configurable API. Refer to the library that provides these exporters for details on how these configuration options works.

You can use the Jupyter configuration files to configure an external exporter. As for any `nbconvert` exporters you can use either the configuration file syntax of `c.MyExporter.config_option=value` or the command line flag form `--MyExporter.config_option=value`.

CHAPTER

ELEVEN

WRITING A CUSTOM `EXPORTER`

Under the hood exporters are python classes that expose a certain interface. Any importable classes that expose this interface can be use as an exporter for nbconvert.

For simplicity we expose basic classes that implement all the relevant methods that you have to subclass and overwrite just the relevant methods to provide a custom exporter. Below we show you the step to create a custom exporter that provides a custom file extension, and a custom template that inserts before and after each markdown cell.

We will lay out files to be ready for Python packaging and distributing on PyPI, although the exact art of Python packaging is beyond the scope of this explanation.

We will use the following layout for our package to expose a custom exporter:

```
mypackage
├── LICENSE.md
├── setup.py
└── mypackage
    ├── __init__.py
    └── templates
        └── test_template.tpl
```

If you wished to create this same directory structure you could use the following commands when you are at the directory under which you wish to build your `mypackage` package:

```
mkdir -p mypackage/mypackage/templates
touch mypackage/LICENSE.md
touch mypackage/setup.py
touch mypackage/mypackage/__init__.py
touch mypackage/mypackage/templates/test_template.tpl
```

Important: You should not publish this package without adding content to your `LICENSE.md` file. For example, `nbconvert` follows the Jupyter Project convention of using a Modified BSD License (also known as New or Revised or 3-Clause BSD). For a guide on picking the right license for your use case, please see choose a license. If you do not specify the license, your code may be unusable by many open source projects.

As you can see the layout is relatively simple, in the case where a template is not needed we would actually have only one file with an Exporter implementation. Of course you can change the layout of your package to have a more fine-grained structure of the subpackage. But lets see what a minimum example looks like.

We are going to write an exporter that:

- exports to html, so we will reuse the built-in html exporter
- changes the file extension to `.test_ext`

```
# file __init__.py
import os
import os.path

from traitlets.config import Config
from nbconvert.exporters.html import HTMLExporter

#-----------------------------------------------------------------------------
# Classes
#-----------------------------------------------------------------------------

class MyExporter(HTMLExporter):
    """
    My custom exporter
    """

    # If this custom exporter should add an entry to the
    # "File -> Download as" menu in the notebook, give it a name here in the
    # `export_from_notebook` class member
    export_from_notebook = "My format"

    def _file_extension_default(self):
        """
        The new file extension is ``.test_ext``
        """
        return '.test_ext'

    @property
    def template_path(self):
        """
        We want to inherit from HTML template, and have template under
        ``./templates/`` so append it to the search path. (see next section)
        """
        return super().template_path+[os.path.join(os.path.dirname(__file__),
↪"templates")]

    def _template_file_default(self):
        """
        We want to use the new template we ship with our library.
        """
        return 'test_template' # full
```

And the template file, that inherits from the html `full` template and prepend/append text to each markdown cell (see Jinja2 docs for template syntax):

```
{% extends "full.tpl" %}

{% block markdowncell -%}


## this is a markdown cell
{{ super() }}
## THIS IS THE END


{% endblock markdowncell %}
```

Assuming you install this package locally, or from PyPI, you can now use:

```
jupyter nbconvert --to mypackage.MyExporter notebook.ipynb
```

CHAPTER

TWELVE

CUSTOMIZING SYNTAX HIGHLIGHTING

Under the hood, nbconvert uses pygments to highlight code. pdf, webpdf and html exporting support changing the highlighting style.

12.1 Using Builtin styles

Pygments has a number of builtin styles available. To use them, we just need to set the style setting in the relevant preprocessor.

To change html and webpdf highlighting export with:

```
jupyter nbconvert --to html notebook.ipynb --CSSHTMLHeaderPreprocessor.style=<name>
```

To change pdf and latex highlighting export with:

```
jupyter nbconvert --to pdf notebook.ipynb --LatexPreprocessor.style=<name>
```

where <name> is the name of the pygments style. Available styles may vary from system to system. You can find all available styles with:

```
pygmentize -L styles
```

from a terminal or

```
from pygments.styles import get_all_styles
print(list(get_all_styles()))
```

from python.

You can preview all the styles from an environment that can display html like jupyter notebook with:

```
from pygments.styles import get_all_styles
from pygments.formatters import Terminal256Formatter
from pygments.lexers import PythonLexer
from pygments import highlight

code = """
import os
def function(test=1):
    if test in [3,4]:
      print(test)
"""
for style in get_all_styles():
```

(continues on next page)

(continued from previous page)

```
    highlighted_code = highlight(code, PythonLexer(),␣
↪Terminal256Formatter(style=style))
    print(f"{style}:\n{highlighted_code}")
```

12.2 Making your own styles

To make your own style you must subclass `pygments.styles.Style`, and then you must register your new style with Pygments using their plugin system. This is explained in detail in the Pygments documentation.

CHAPTER

THIRTEEN

ARCHITECTURE OF NBCONVERT

This is a high-level outline of the basic workflow, structures and objects in nbconvert. Specifically, this exposition has a two-fold goal:

1. to alert you to the affordances available for customisation or direct contributions
2. to provide a map of where and when different events occur, which should aid in tracking down bugs.

13.1 A detailed pipeline exploration

Nbconvert takes in a notebook, which is a JSON object, and operates on that object.

This can include operations that take a notebook and return a notebook. For example, that operation could be to execute the notebook as though it were a continuous script; if it were executed `--in-place` then it would overwrite the current notebook. Or it could be that we wish to systematically alter the notebook, for example by clearing all output cells. Format agnostic operations on cell content that do not violate the nbformat spec can be interpreted as a notebook to notebook conversion step; such operations can be performed as part of the preprocessing step.

But often we want to have the notebook's structured content in a different format. Importantly, in many cases the structure of the notebook should be reflected in the structure of the output, adapted to the output's format. For that purpose, the original JSON structure of the document is crucial scaffolding needed to support this kind of structured output. In order to maintain structure, it can be useful to apply our conversion programmatically on the structure itself. To do so, when converting to formats other than the notebook, we use the jinja templating engine.

The basic unit of structure in a notebook is the cell. Accordingly, since our templating engine is capable of expressing structure, the basic unit in our templates will often be specified at the cell level. Each cell has a certain type; the three most important cell types for our purposes are code, markdown, and raw NbConvert. Code cells can be split further into their input and their output. Operations can also occur separately on input and output and their respective subcomponents. Markdown cells and raw NbConvert cells do not have analogous substructure.

The template's structure then can be seen as a mechanism for selecting content on which to operate. Because the template operates on individual cells, this has some upsides and drawbacks. One upside is that this allows the template to have access to the individual cell's metadata, which enables intelligently transforming the appropriate content. The transformations occur as a series of replacement rules and filters. For many purposes these filters take the form of external calls to pandoc, which is a utility for converting between many different document formats. One downside is that this makes operations that require global coördination (e.g., cross referencing across cells) somewhat challenging to implement as filters inside templates.

Note that all that we've described is happening in memory. This is crucial in order to ensure that this functionality is available when writing files is more challenging. Nonetheless, the reason for using nbconvert almost always involves producing some kind of output file. We take the in-memory object and write a file appropriate for the output type.

The entirety of heretofore described process can be described as part of an `Exporter`. `Exporters` often involves `Preprocessors`, `filters`, `templates` and `Writers`. These classes and functions are described in greater

detail below.

Finally, one can apply a `Postprocessor` after the writing has occurred. For example, it is common when converting to slides to start a webserver and open a browser window with the newly created document (`--to slides --post serve`).

13.2 Classes

13.2.1 Exporters

The primary class in nbconvert is the *`Exporter`*. Exporters encapsulate the operation of turning a notebook into another format. There is one Exporter for each format supported in nbconvert. The first thing an Exporter does is load a notebook, usually from a file via `nbformat`. Most of what a typical Exporter does is select and configure preprocessors, filters, and templates. If you want to convert notebooks to additional formats, a new Exporter is probably what you are looking for.

See also:

Writing a custom Exporter

Once the notebook is loaded, it is preprocessed...

13.2.2 Preprocessors

A *`Preprocessor`* is an object that transforms the content of the notebook to be exported. The result of a preprocessor being applied to a notebook is always a notebook. These operations include re-executing the cells, stripping output, removing bundled outputs to separate files, etc. If you want to add operations that modify a notebook before exporting, a preprocessor is the place to start.

See also:

Custom Preprocessors

Once a notebook is preprocessed, it's time to convert the notebook into the destination format.

13.2.3 Templates

Most Exporters in nbconvert are a subclass of *`TemplateExporter`*, which make use of jinja to render a notebook into the destination format.

Nbconvert templates can be selected from the command line with the `--template` option. For example, to use the `reveal` template with the HTML exporter

```
jupyter nbconvert <path-to-notebook> --to html --template reveal
```

Note: Since version 6.0, The HTML exporter defaults to the `lab` template which produces a DOM structure corresponding to the notebook component in JupyterLab.

To produce HTML corresponding to the looks of the classic notebook, one can use the `classic` template by passing `--template classic` to the command line.

The nbconvert template system has been completely revamped with nbconvert 6.0 to allow for greater extensibility. Nbconvert templates can now be installed as third-party packages and are automatically picked up by nbconvert.

For more details about how to create custom templates, check out the *Creating Custom Templates for nbconvert* section of the documentation.

13.2.4 Filters

Filters are Python callables which take something (typically text) as an input, and produce a text output. If you want to perform custom transformations of particular outputs, a filter may be the way to go.

The following code snippet is an excert from the main default template of the HTML export. The displayed block determines how text output on `stdout` is displayed in HTML.

```
{% block stream_stdout -%}
<div class="output_subarea output_stream output_stdout output_text">
<pre>
{{- output.text | ansi2html -}}
</pre>
</div>
{%- endblock stream_stdout %}
```

In the `{{- output.text | ansi2html -}}` bit, we invoke the `ansi2html` filter to transform the text output.

Typically, filters are pure functions. However, filters that require some configuration, may be implemented as Configurable classes.

See also:

- *Creating Custom Templates for nbconvert*
- Filters

Once it has passed through the template, an Exporter is done with the notebook, and returns the file data.

At this point, we have the file data as text or bytes and we can decide where it should end up. When you are using nbconvert as a library, as opposed to the command-line application, this is typically where you would stop, take your exported data, and go on your way.

13.2.5 Writers

A *Writer* takes care of writing the resulting file(s) where they should end up. There are two basic Writers in nbconvert:

1. stdout - writes the result to stdout (for pipe-style workflows)
2. Files (default) - writes the result to the filesystem

Once the output is written, nbconvert has done its job.

13.2.6 Postprocessors

A *Postprocessor* is something that runs after everything is exported and written to the filesystem. The only postprocessor in nbconvert at this point is the *ServePostProcessor*, which is used for serving reveal.js HTML slideshows.

CHAPTER

FOURTEEN

PYTHON API FOR WORKING WITH NBCONVERT

Contents:

14.1 NbConvertApp

See also:

Configuration options Configurable options for the nbconvert application

class nbconvert.nbconvertapp.**NbConvertApp**(***kwargs*)

Application used to convert from notebook file type (`*.ipynb`)

init_notebooks()

Construct the list of notebooks.

If notebooks are passed on the command-line, they override (rather than add) notebooks specified in config files. Glob each notebook to replace notebook patterns with filenames.

convert_notebooks()

Convert the notebooks in the self.notebook traitlet

convert_single_notebook(*notebook_filename*, *input_buffer=None*)

Convert a single notebook.

Performs the following steps:

1. Initialize notebook resources
2. Export the notebook to a particular format
3. Write the exported notebook to file
4. (Maybe) postprocess the written file

Parameters

- **notebook_filename** (*str*) –
- **input_buffer** – If input_buffer is not None, conversion is done and the buffer is used as source into a file basenamed by the notebook_filename argument.

init_single_notebook_resources(*notebook_filename*)

Step 1: Initialize resources

This initializes the resources dictionary for a single notebook.

Returns

resources dictionary for a single notebook that MUST include the following keys:

- config_dir: the location of the Jupyter config directory
- unique_key: the notebook name
- output_files_dir: a directory where output files (not including the notebook itself) should be saved

Return type dict

export_single_notebook(*notebook_filename*, *resources*, *input_buffer=None*)
Step 2: Export the notebook

Exports the notebook to a particular format according to the specified exporter. This function returns the output and (possibly modified) resources from the exporter.

Parameters

- **notebook_filename** (*str*) – name of notebook file.
- **resources** (*dict*) –
- **input_buffer** – readable file-like object returning unicode. if not None, notebook_filename is ignored

Returns

- *output*
- *dict* – resources (possibly modified)

write_single_notebook(*output*, *resources*)
Step 3: Write the notebook to file

This writes output from the exporter to file using the specified writer. It returns the results from the writer.

Parameters

- **output** –
- **resources** (*dict*) – resources for a single notebook including name, config directory and directory to save output

Returns results from the specified writer output of exporter

Return type file

postprocess_single_notebook(*write_results*)
Step 4: Post-process the written file

Only used if a postprocessor has been specified. After the converted notebook is written to a file in Step 3, this post-processes the notebook.

14.2 Exporters

See also:

Configuration options Configurable options for the nbconvert application

nbconvert.exporters.**export**(*exporter*, *nb*, ***kw*)
Export a notebook object using specific exporter class.

Parameters

- **exporter** (*Exporter* class or instance) – Class or instance of the exporter that should be used. If the method initializes its own instance of the class, it is ASSUMED that the class type provided exposes a constructor (__init__) with the same signature as the base Exporter class.
- **nb** (nbformat.NotebookNode) – The notebook to export.
- **config** (*config (optional, keyword arg)*) – User configuration instance.
- **resources** (*dict (optional, keyword arg)*) – Resources used in the conversion process.

Returns

output [str] The resulting converted notebook.

resources [dictionary] Dictionary of resources used prior to and during the conversion process.

Return type tuple

nbconvert.exporters.**get_exporter**(*name*, *config={}*)
Given an exporter name or import path, return a class ready to be instantiated

Raises ExporterName if exporter is not found or ExporterDisabledError if not enabled

nbconvert.exporters.**get_export_names**(*config={}*)
Return a list of the currently supported export targets

Exporters can be found in external packages by registering them as an nbconvert.exporter entrypoint.

14.2.1 Exporter base classes

class nbconvert.exporters.**Exporter**(***kwargs*)
Class containing methods that sequentially run a list of preprocessors on a NotebookNode object and then return the modified NotebookNode object and accompanying resources dict.

__init__(*config=None*, ***kw*)
Public constructor

Parameters

- **config** (traitlets.config.Config) – User configuration instance.
- ****kw** – Additional keyword arguments passed to parent __init__

from_notebook_node(*nb*, *resources=None*, ***kw*)
Convert a notebook from a notebook node instance.

Parameters

- **nb** (nbformat.NotebookNode) – Notebook node (dict-like with attr-access)
- **resources** (*dict*) – Additional resources that can be accessed read/write by preprocessors and filters.
- ****kw** – Ignored

from_filename(*filename: str*, *resources: Optional[dict] = None*, ***kw*)
Convert a notebook from a notebook file.

Parameters

- **filename** (*str*) – Full filename of the notebook file to open and convert.

- **resources** (*dict*) – Additional resources that can be accessed read/write by preprocessors and filters.
- ****kw** – Ignored

from_file (*file_stream*, *resources=None*, ***kw*)
Convert a notebook from a notebook file.

Parameters

- **file_stream** (*file-like object*) – Notebook file-like object to convert.
- **resources** (*dict*) – Additional resources that can be accessed read/write by preprocessors and filters.
- ****kw** – Ignored

register_preprocessor (*preprocessor*, *enabled=False*)
Register a preprocessor. Preprocessors are classes that act upon the notebook before it is passed into the Jinja templating engine. preprocessors are also capable of passing additional information to the Jinja templating engine.

Parameters

- **preprocessor** (*Preprocessor*) – A dotted module name, a type, or an instance
- **enabled** (*bool*) – Mark the preprocessor as enabled

class nbconvert.exporters.**TemplateExporter** (***kwargs*)
Exports notebooks into other file formats. Uses Jinja 2 templating engine to output new formats. Inherit from this class if you are creating a new template type along with new filters/preprocessors. If the filters/ preprocessors provided by default suffice, there is no need to inherit from this class. Instead, override the template_file and file_extension traits via a config file.

Filters available by default for templates:

- add_anchor
- add_prompts
- ansi2html
- ansi2latex
- ascii_only
- citation2latex
- comment_lines
- convert_pandoc
- escape_latex
- filter_data_type
- get_lines
- get_metadata
- highlight2html
- highlight2latex
- html2text
- indent

- ipython2python
- json_dumps
- markdown2asciidoc
- markdown2html
- markdown2latex
- markdown2rst
- path2url
- posix_path
- prevent_list_blocks
- strip_ansi
- strip_dollars
- strip_files_prefix
- strip_trailing_newline
- wrap_text

__init__(*config=None*, ***kw*)
Public constructor

Parameters

- **config** (*config*) – User configuration instance.
- **extra_loaders** (*list[of Jinja Loaders]*) – ordered list of Jinja loader to find templates. Will be tried in order before the default FileSystem ones.
- **template_file** (*str (optional, kw arg)*) – Template to use when exporting.

from_notebook_node(*nb*, *resources=None*, ***kw*)
Convert a notebook from a notebook node instance.

Parameters

- **nb** (nbformat.NotebookNode) – Notebook node
- **resources** (*dict*) – Additional resources that can be accessed read/write by preprocessors and filters.

from_filename(*filename: str*, *resources: Optional[dict] = None*, ***kw*)
Convert a notebook from a notebook file.

Parameters

- **filename** (*str*) – Full filename of the notebook file to open and convert.
- **resources** (*dict*) – Additional resources that can be accessed read/write by preprocessors and filters.
- ****kw** – Ignored

from_file(*file_stream*, *resources=None*, ***kw*)
Convert a notebook from a notebook file.

Parameters

- **file_stream** (*file-like object*) – Notebook file-like object to convert.

- **resources** (*dict*) – Additional resources that can be accessed read/write by preprocessors and filters.
- ****kw** – Ignored

register_preprocessor(*preprocessor*, *enabled=False*)
Register a preprocessor. Preprocessors are classes that act upon the notebook before it is passed into the Jinja templating engine. preprocessors are also capable of passing additional information to the Jinja templating engine.

Parameters

- **preprocessor** (*Preprocessor*) – A dotted module name, a type, or an instance
- **enabled** (*bool*) – Mark the preprocessor as enabled

register_filter(*name*, *jinja_filter*)
Register a filter. A filter is a function that accepts and acts on one string. The filters are accessible within the Jinja templating engine.

Parameters

- **name** (*str*) – name to give the filter in the Jinja engine
- **filter** (*filter*) –

14.2.2 Specialized exporter classes

The *NotebookExporter* inherits directly from *Exporter*, while the other exporters listed here inherit either directly or indirectly from *TemplateExporter*.

class nbconvert.exporters.**NotebookExporter**(***kwargs*)
Exports to an IPython notebook.

This is useful when you want to use nbconvert's preprocessors to operate on a notebook (e.g. to execute it) and then write it back to a notebook file.

class nbconvert.exporters.**HTMLExporter**(***kwargs*)
Exports a basic HTML document. This exporter assists with the export of HTML. Inherit from it if you are writing your own HTML template and need custom preprocessors/filters. If you don't need custom preprocessors/filters, just change the 'template_file' config option.

class nbconvert.exporters.**SlidesExporter**(***kwargs*)
Exports HTML slides with reveal.js

class nbconvert.exporters.**LatexExporter**(***kwargs*)
Exports to a Latex template. Inherit from this class if your template is LaTeX based and you need custom transformers/filters. If you don't need custom transformers/filters, just change the 'template_file' config option. Place your template in the special "/latex" subfolder of the "../templates" folder.

class nbconvert.exporters.**MarkdownExporter**(***kwargs*)
Exports to a markdown document (.md)

class nbconvert.exporters.**PDFExporter**(***kwargs*)
Writer designed to write to PDF files.

This inherits from *LatexExporter*. It creates a LaTeX file in a temporary directory using the template machinery, and then runs LaTeX to create a pdf.

class nbconvert.exporters.**WebPDFExporter**(***kwargs*)
Writer designed to write to PDF files.

This inherits from *HTMLExporter*. It creates the HTML using the template machinery, and then run pyppeteer to create a pdf.

class nbconvert.exporters.**PythonExporter**(***kwargs*)
Exports a Python code file.

class nbconvert.exporters.**RSTExporter**(***kwargs*)
Exports reStructuredText documents.

14.3 Preprocessors

See also:

Configuration options Configurable options for the nbconvert application

class nbconvert.preprocessors.**Preprocessor**(***kwargs*)
A configurable preprocessor

Inherit from this class if you wish to have configurability for your preprocessor.

Any configurable traitlets this class exposed will be configurable in profiles using c.SubClassName.attribute = value

You can overwrite *preprocess_cell()* to apply a transformation independently on each cell or *preprocess()* if you prefer your own logic. See corresponding docstring for information.

Disabled by default and can be enabled via the config by 'c.YourPreprocessorName.enabled = True'

__init__(***kw*)
Public constructor

Parameters

- **config** (*Config*) – Configuration file structure
- ****kw** – Additional keyword arguments passed to parent

preprocess(*nb*, *resources*)
Preprocessing to apply on each notebook.

Must return modified nb, resources.

If you wish to apply your preprocessing to each cell, you might want to override preprocess_cell method instead.

Parameters

- **nb** (*NotebookNode*) – Notebook being converted
- **resources** (*dictionary*) – Additional resources used in the conversion process. Allows preprocessors to pass variables into the Jinja engine.

preprocess_cell(*cell*, *resources*, *index*)
Override if you want to apply some preprocessing to each cell. Must return modified cell and resource dictionary.

Parameters

- **cell** (*NotebookNode cell*) – Notebook cell being processed
- **resources** (*dictionary*) – Additional resources used in the conversion process. Allows preprocessors to pass variables into the Jinja engine.
- **index** (*int*) – Index of the cell being processed

14.3.1 Specialized preprocessors

Converting and extracting figures

class nbconvert.preprocessors.**ConvertFiguresPreprocessor**(***kwargs*)
Converts all of the outputs in a notebook from one format to another.

class nbconvert.preprocessors.**SVG2PDFPreprocessor**(***kwargs*)
Converts all of the outputs in a notebook from SVG to PDF.

class nbconvert.preprocessors.**ExtractOutputPreprocessor**(***kwargs*)
Extracts all of the outputs from the notebook file. The extracted outputs are returned in the 'resources' dictionary.

Converting text

class nbconvert.preprocessors.**LatexPreprocessor**(***kwargs*)
Preprocessor for latex destined documents.

Mainly populates the `latex` key in the resources dict, adding definitions for pygments highlight styles.

class nbconvert.preprocessors.**HighlightMagicsPreprocessor**(***kwargs*)
Detects and tags code cells that use a different languages than Python.

Metadata and header control

class nbconvert.preprocessors.**ClearMetadataPreprocessor**(***kwargs*)
Removes all the metadata from all code cells in a notebook.

class nbconvert.preprocessors.**CSSHTMLHeaderPreprocessor**(***kwargs*)
Preprocessor used to pre-process notebook for HTML output. Adds IPython notebook front-end CSS and Pygments CSS to HTML output.

Removing cells, inputs, and outputs

class nbconvert.preprocessors.**ClearOutputPreprocessor**(***kwargs*)
Removes the output from all code cells in a notebook.

class nbconvert.preprocessors.**RegexRemovePreprocessor**(***kwargs*)
Removes cells from a notebook that match one or more regular expression.

For each cell, the preprocessor checks whether its contents match the regular expressions in the `patterns` traitlet which is a list of unicode strings. If the contents match any of the patterns, the cell is removed from the notebook.

To modify the list of matched patterns, modify the patterns traitlet. For example, execute the following command to convert a notebook to html and remove cells containing only whitespace:

```
jupyter nbconvert --RegexRemovePreprocessor.patterns="['\s*\Z']" mynotebook.ipynb
```

The command line argument sets the list of patterns to `'\s*\Z'` which matches an arbitrary number of whitespace characters followed by the end of the string.

See https://regex101.com/ for an interactive guide to regular expressions (make sure to select the python flavor). See https://docs.python.org/library/re.html for the official regular expression documentation in python.

class nbconvert.preprocessors.**TagRemovePreprocessor**(***kwargs*)
Removes inputs, outputs, or cells from a notebook that have tags that designate they are to be removed prior to exporting the notebook.

remove_cell_tags removes cells tagged with these values

remove_all_outputs_tags removes entire output areas on cells tagged with these values

remove_single_output_tags removes individual output objects on outputs tagged with these values

remove_input_tags removes inputs tagged with these values

Executing Notebooks

class nbconvert.preprocessors.**ExecutePreprocessor**(***kwargs*)
Executes all the cells in a notebook

preprocess(*nb*, *resources=None*, *km=None*)
Preprocess notebook executing each code cell.

The input argument *nb* is modified in-place.

Parameters

- **nb** (*NotebookNode*) – Notebook being executed.
- **resources** (*dictionary (optional)*) – Additional resources used in the conversion process. For example, passing `{'metadata': {'path': run_path}}` sets the execution path to `run_path`.
- **km** (*KernelManager (optional)*) – Optional kernel manager. If none is provided, a kernel manager will be created.

Returns

- **nb** (*NotebookNode*) – The executed notebook.
- **resources** (*dictionary*) – Additional resources used in the conversion process.

class nbconvert.preprocessors.**CellExecutionError**(*traceback: str*, *ename: str*, *evalue: str*)
Custom exception to propagate exceptions that are raised during notebook execution to the caller. This is mostly useful when using nbconvert as a library, since it allows to deal with failures gracefully.

nbconvert.preprocessors.**coalesce_streams**(*cell*, *resources*, *index*)
Merge consecutive sequences of stream output into single stream to prevent extra newlines inserted at flush calls

Parameters

- **cell** (*NotebookNode cell*) – Notebook cell being processed
- **resources** (*dictionary*) – Additional resources used in the conversion process. Allows transformers to pass variables into the Jinja engine.
- **index** (*int*) – Index of the cell being processed

14.4 Filters

Filters are for use with the *TemplateExporter* exporter. They provide a way for you transform notebook contents to a particular format depending on the template you are using. For example, when converting to HTML, you would want to use the *ansi2html()* function to convert ANSI colors (from e.g. a terminal traceback) to HTML colors.

See also:

Exporters API documentation for the various exporter classes

nbconvert.filters.**add_anchor**(*html*, *anchor_link_text='¶'*)
Add an id and an anchor-link to an html header

For use on markdown headings

nbconvert.filters.**add_prompts**(*code*, *first='>>> '*, *cont='... '*)
Add prompts to code snippets

nbconvert.filters.**ansi2html**(*text*)
Convert ANSI colors to HTML colors.

Parameters **text** (*unicode*) – Text containing ANSI colors to convert to HTML

nbconvert.filters.**ansi2latex**(*text*)
Convert ANSI colors to LaTeX colors.

Parameters **text** (*unicode*) – Text containing ANSI colors to convert to LaTeX

nbconvert.filters.**ascii_only**(*s*)
ensure a string is ascii

nbconvert.filters.**citation2latex**(*s*)
Parse citations in Markdown cells.

This looks for HTML tags having a data attribute names `data-cite` and replaces it by the call to LaTeX cite command. The transformation looks like this:

```
<cite data-cite="granger">(Granger, 2013)</cite>
```

Becomes

```
\cite{granger}
```

Any HTML tag can be used, which allows the citations to be formatted in HTML in any manner.

nbconvert.filters.**comment_lines**(*text*, *prefix='# '*)
Build a Python comment line from input text.

Parameters

- **text** (*str*) – Text to comment out.
- **prefix** (*str*) – Character to append to the start of each line.

nbconvert.filters.**escape_latex**(*text*)
Escape characters that may conflict with latex.

Parameters **text** (*str*) – Text containing characters that may conflict with Latex

class nbconvert.filters.**DataTypeFilter**(***kwargs*)
Returns the preferred display format

nbconvert.filters.**get_lines**(*text*, *start=None*, *end=None*)
Split the input text into separate lines and then return the lines that the caller is interested in.

Parameters

- **text** (*str*) – Text to parse lines from.
- **start** (*int, optional*) – First line to grab from.
- **end** (*int, optional*) – Last line to grab from.

nbconvert.filters.**convert_pandoc**(*source*, *from_format*, *to_format*, *extra_args=None*)
Convert between any two formats using pandoc.

This function will raise an error if pandoc is not installed. Any error messages generated by pandoc are printed to stderr.

Parameters

- **source** (*string*) – Input string, assumed to be valid in from_format.
- **from_format** (*string*) – Pandoc format of source.
- **to_format** (*string*) – Pandoc format for output.

Returns **out** – Output as returned by pandoc.

Return type string

class nbconvert.filters.**Highlight2HTML**(***kwargs*)

class nbconvert.filters.**Highlight2Latex**(***kwargs*)

nbconvert.filters.**html2text**(*element*)
extract inner text from html

Analog of jQuery's $(element).text()

nbconvert.filters.**indent**(*instr*, *nspaces=4*, *ntabs=0*, *flatten=False*)
Indent a string a given number of spaces or tabstops.

indent(str,nspaces=4,ntabs=0) -> indent str by ntabs+nspaces.

Parameters

- **instr** (*basestring*) – The string to be indented.
- **nspaces** (*int (default: 4)*) – The number of spaces to be indented.
- **ntabs** (*int (default: 0)*) – The number of tabs to be indented.
- **flatten** (*bool (default: False)*) – Whether to scrub existing indentation. If True, all lines will be aligned to the same indentation. If False, existing indentation will be strictly increased.

Returns **str|unicode**

Return type string indented by ntabs and nspaces.

nbconvert.filters.**ipython2python**(*code*)
Transform IPython syntax to pure Python syntax

Parameters **code** (*str*) – IPython code, to be transformed to pure Python

nbconvert.filters.**markdown2html**(*source*)
Convert a markdown string to HTML using mistune

nbconvert.filters.**markdown2latex**(*source*, *markup='markdown'*, *extra_args=None*)
Convert a markdown string to LaTeX via pandoc.

This function will raise an error if pandoc is not installed. Any error messages generated by pandoc are printed to stderr.

Parameters

- **source** (*string*) – Input string, assumed to be valid markdown.
- **markup** (*string*) – Markup used by pandoc's reader default : pandoc extended markdown (see https://pandoc.org/README.html#pandocs-markdown)

Returns **out** – Output as returned by pandoc.

Return type string

nbconvert.filters.**markdown2rst**(*source*, *extra_args=None*)
Convert a markdown string to ReST via pandoc.

This function will raise an error if pandoc is not installed. Any error messages generated by pandoc are printed to stderr.

Parameters **source** (*string*) – Input string, assumed to be valid markdown.

Returns **out** – Output as returned by pandoc.

Return type string

nbconvert.filters.**path2url**(*path*)
Turn a file path into a URL

nbconvert.filters.**posix_path**(*path*)
Turn a path into posix-style path/to/etc

Mainly for use in latex on Windows, where native Windows paths are not allowed.

nbconvert.filters.**prevent_list_blocks**(*s*)
Prevent presence of enumerate or itemize blocks in latex headings cells

nbconvert.filters.**strip_ansi**(*source*)
Remove ANSI escape codes from text.

Parameters **source** (*str*) – Source to remove the ANSI from

nbconvert.filters.**strip_dollars**(*text*)
Remove all dollar symbols from text

Parameters **text** (*str*) – Text to remove dollars from

nbconvert.filters.**strip_files_prefix**(*text*)
Fix all fake URLs that start with `files/`, stripping out the `files/` prefix. Applies to both urls (for html) and relative paths (for markdown paths).

Parameters **text** (*str*) – Text in which to replace 'src="files/real...' with 'src="real...'

nbconvert.filters.**wrap_text**(*text*, *width=100*)
Intelligently wrap text. Wrap text without breaking words if possible.

Parameters

- **text** (*str*) – Text to wrap.
- **width** (*int, optional*) – Number of characters to wrap to, default 100.

14.5 Writers

See also:

Configuration options Configurable options for the nbconvert application

class nbconvert.writers.**WriterBase**(***kwargs*)
Consumes output from nbconvert export...() methods and writes to a useful location.

__init__(*config=None*, ***kw*)
Constructor

write(*output*, *resources*, ***kw*)
Consume and write Jinja output.

Parameters

- **output** (*string*) – Conversion results. This string contains the file contents of the converted file.
- **resources** (*dict*) – Resources created and filled by the nbconvert conversion process. Includes output from preprocessors, such as the extract figure preprocessor.

14.5.1 Specialized writers

class nbconvert.writers.**DebugWriter**(***kwargs*)
Consumes output from nbconvert export...() methods and writes useful debugging information to the stdout. The information includes a list of resources that were extracted from the notebook(s) during export.

class nbconvert.writers.**FilesWriter**(***kwargs*)
Consumes nbconvert output and produces files.

class nbconvert.writers.**StdoutWriter**(***kwargs*)
Consumes output from nbconvert export...() methods and writes to the stdout stream.

14.6 Postprocessors

See also:

Configuration options Configurable options for the nbconvert application

class nbconvert.postprocessors.**PostProcessorBase**(***kwargs*)

postprocess(*input*)
Post-process output from a writer.

14.6.1 Specialized postprocessors

class nbconvert.postprocessors.**ServePostProcessor**(***kwargs*)
Post processor designed to serve files

Proxies reveal.js requests to a CDN if no local reveal.js is present

postprocess(*input*)
Serve the build directory with a webserver.

CHAPTER

FIFTEEN

MAKING AN NBCONVERT RELEASE

This document guides a contributor through creating a release of `nbconvert`.

15.1 Assign all merged PRs to milestones

Go to GitHub and assign all PRs that have been merged to milestones. This will be helpful when you update the changelog. If you go to this GitHub page you will find all the PRs that currently have no milestones.

15.2 Gather all PRs related to milestone

ghpro can be used to extract the pull requests by call the following from nbconvert directory (will ask for an API token the first time):

```
github-stats --milestone=$VERSION --since-tag $LAST_VERSION --links
```

15.3 Manually categorize tickets

Group the tickets by these general categories (or others if they are relevant). This usually a manual processes to evaluate the changes in each PR.

1. New Features
2. Deprecations
3. Fixing Problems
4. Testing, Docs, and Builds

15.4 Collect major changes

From the tickets write up any major features / changes that deserve a paragraph to describe how they work.

15.5 Update docs/source/changelog.rst

Copy these changes with the new version to the top of changelog.rst. Prior release changelogs can be used to pick formatting of the message.

15.6 Check installed tools

Review `CONTRIBUTING.md`, particularly the testing and release sections.

15.7 Clean the repository

You can remove all non-tracked files with:

```
git clean -xfdi
```

This would ask you for confirmation before removing all untracked files.

Make sure the `dist/` and `build/` folders are clean and avoid stale builds from previous attempts.

15.8 Create the release

1. Update the *changelog* to account for all the PRs assigned to this milestone.
2. Update version number in `notebook/_version.py` and remove `.dev` from dev_info. Note that the version may already be on the dev version of the number you're releasing.
3. Commit and tag the release with the current version number:

```
git commit -am "release $VERSION"
git tag $VERSION
```

4. You are now ready to build the `sdist` and `wheel`:

```
python setup.py sdist
python setup.py bdist_wheel
```

5. You can now test the `wheel` and the `sdist` locally before uploading to PyPI. Make sure to use twine to upload the archives over SSL.

```
twine upload dist/*
```

6. The conda-forge bot will automatically add a PR on your behalf to the nbconvert-feedstock repo. You may want to review this PR to ensure conda-forge will be updated cleanly.

15.9 Release the new version

Push directly on master, including –tags separately

```
git push upstream
git push upstream --tags
```

15.10 Return to development state

If all went well, change the `notebook/_version.py` back by adding the `.dev` suffix and moving the version forward to the next patch release number.

15.11 Email googlegroup with update letter

Make sure to email jupyter@googlegroups.com with the subject line of "[ANN] NBConvert $VERSION – ..." and include at least the significant changes, contributors, and individual PR notes (if not many significant changes).

CHAPTER

SIXTEEN

CHANGES IN NBCONVERT

16.1 6.0.0

The following authors and reviewers contributed the changes for this release – Thanks you all!

- Ayaz Salikhov
- bnables
- Bo
- David Brochart
- David Cortés
- Eric Wieser
- Florian Rathgeber
- Ian Allison
- James Wilshaw
- Jeremy Tuloup
- Joel Ostblom
- Jon Bannister
- Jonas Drotleff
- Josh Devlin
- Karthikeyan Singaravelan
- Kerwin.Sun
- letmerecall
- Luciano Resende
- Lumír 'Frenzy' Balhar
- Maarten A. Breddels
- Maarten Breddels
- Marcel Stimberg
- Matthew Brett
- Matthew Seal
- Matthias Bussonnier

- Matthias Geier
- Miro Hrončok
- Phil Austin
- Praveen Batra
- Ruben Di Battista
- Ruby Werman
- Sang-Yun Oh
- Sergey Kizunov
- Sundar
- Sylvain Corlay
- telamonian
- Thomas Kluyver
- Thomas Ytterdal
- Tyler Makaro
- Yu-Cheng (Henry) Huang

16.1.1 Significant changes

Nbconvert 6.0 is a major release of nbconvert which includes many significant changes.

- Python 2 support was dropped. Currently Python 3.6-3.8 is supported and tested by nbconvert. However, nbconvert 6.0 provides limited support for Python 3.6. nbconvert 6.1 will drop support for Python 3.6. Limited support means we will test and run CI on Python 3.6.12 or higher. Issues that are found only affecting Python 3.6 are not guaranteed to be fixed. We recommend all users of nbconvert use Python 3.7 and higher.
- Unlike previous versions, nbconvert 6.0 relies on the nbclient package for the execute preprocessor, which allows for asynchronous kernel requests.
- Nbconvert 6.0 includes a new "webpdf" exporter, which renders notebooks in pdf format through a headless web browser, so that complex outputs such as HTML tables, or even widgets are rendered in the same way as with the HTML exporter and a web browser.
- The default template applied when exporting to HTML now produces the same DOM structure as JupyterLab, and is styled using JupyterLab's CSS. The pygments theme in use mimics JupyterLab's codemirror mode with the same CSS variables, so that custom JupyterLab themes could be applied. The classic notebook styling can still be enabled with

```
jupyter nbconvert --to html --template classic
```

- Nbconvert 6.0 includes a new system for creating custom templates, which can now be installed as packages. A custom "foobar" template is installed in Jupyter's data directory under `nbconvert/templates` and has the form of a directory containing all resources. Templates specify their base template as well as other configuration parameters in a `conf.json` at the root of the template directory.
- The "slideshow" template now makes use of RevealJS version 4. It can now be used with the HTML exporter with

```
jupyter nbconvert --to html --template reveal
```

The `--to slides` exporter is still supported for convenience.

- Inkscape 1.0 is now supported, which had some breaking changes that prevented 5.x versions of nbconvert from converting documents on some systems that updated.

16.1.2 Remaining changes

We merged 105 pull requests! Rather than enumerate all of them we'll link to the github page which contains the many smaller impact improvements.

The full list can be seen on GitHub

16.2 5.6.1

The following authors and reviewers contributed the changes for this release – Thanks you all!

- Charles Frye
- Chris Holdgraf
- Felipe Rodrigues
- Gregor Sturm
- Jim
- Kerwin Sun
- Ryan Beesley
- Matthew Seal
- Matthias Geier
- thuy-van
- Tyler Makaro

16.2.1 Significant Changes

RegExRemove applies to all cells

RegExRemove preprocessor now removes cells regardless of cell outputs. Before this only cells that had outputs were filtered.

16.2.2 Comprehensive notes

New Features

- Add support for alt tags for jpeg and png images PR #1112:
- Allow HTML header anchor text to be HTML PR #1101:
- Change RegExRemove to remove code cells with output PR #1095:
- Added cell tag data attributes to HTML exporter PR #1090: and PR #1089:

Fixing Problems

- Update svg2pdf.py to search the PATH for inkscape PR #1115:
- Fix latex dependencies installation command for Ubuntu systems PR #1109:

Testing, Docs, and Builds

- Added Circle CI builds for documentation PR #1114: PR #1120:, and PR #1116:
- Fix typo in argument name in docstring (TagRemovePreprocessor) PR #1103:
- Changelog typo fix PR #1100:
- Updated API page for TagRemovePreprocessor and TemplateExporter PR #1088:
- Added remove_input_tag traitlet to the docstring PR #1088:

16.3 5.6

The following 24 authors and reviewers contributed 224 commits – Thank you all!

- 00Kai0
- Aidan Feldman
- Alex Rudy
- Alexander Kapshuna
- Alexander Rudy
- amniskin
- Carol Willing
- Dustin H
- Hsiaoming Yang
- imtsuki
- Jessica B. Hamrick
- KrokodileDandy
- Kunal Marwaha
- Matthew Seal
- Matthias Geier
- Miro Hrončok
- M Pacer
- Nils Japke
- njapke
- Sebastian Führ
- Sylvain Corlay
- Tyler Makaro

- Valery M
- Wayne Witzel

The full list of changes they made can be seen on GitHub

16.3.1 Significant Changes

Jupter Client Pin

The `jupyter_client` dependency is now pinned to `>5.3.1`. This is done to support the *Parallel NBConvert* below, and future versions may require interface changes from that version.

Parallel NBConvert

NBConvert `--execute` can now be run in parallel via threads, multiprocessing, or async patterns! This means you can now parallelize nbconvert via a bash loop, or a python concurrency pattern and it should be able to execute those notebooks in parallel.

Kernels have varying support for safe concurrent execution. The ipython kernel (ipykernel version 1.5.2 and higher) should be safe to run concurrently using Python 3. However, the Python 2 ipykernel does not always provide safe concurrent execution and sometimes fails with a socket bind exception. Unlike ipykernel which is maintained by the project, other community-maintained kernels may have varying support for concurrent execution, and these kernels were not tested heavily.

Issues for nbconvert can be viewed here: PR #1018:, and PR #1017:

Execute Loop Rewrite

This release completely rewrote the execution loop responsible for monitoring kernel messages until cell execution is completed. This removes an error where kernel messages could be dropped if too many were posted too quickly. Furthermore, the change means that messages are not buffered. Now, messages can be logged immediately rather than waiting for the cell to terminate.

See PR #994: for exact code changes if you're curious.

16.3.2 Comprehensive notes

New Features

- Make a default global location for custom user templates PR #1028:
- Parallel execution improvements PR #1018:, and PR #1017:
- Added `store_history` option to `preprocess_cell` and `run_cell` PR #1055:
- Simplify the function signature for preprocess() PR #1042:
- Set flag to not always stop kernel execution on errors PR #1040:
- `setup_preprocessor` passes kwargs to `start_new_kernel` PR #1021:

Fixing Problems

- Very fast stream outputs no longer drop some messages PR #994:
- LaTeX errors now properly raise exceptions PR #1053:
- Improve template whitespacing PR #1076:
- Fixes for character in LaTeX exports and filters PR #1068:, PR #1039:, PR #1024:, and PR #1077:
- Mistune pinned in preparation for 2.0 release PR #1074:
- Require mock only on Python 2 PR #1060: and PR #1011:
- Fix selection of mimetype when converting to HTML PR #1036:
- Correct a few typos PR #1029:
- Update `export_from_notebook` names PR #1027:
- Dedenting html in ExtractOutputPreprocessor PR #1023:
- Fix backwards incompatibility with markdown2html PR #1022:
- Fixed html image tagging PR #1013:
- Remove unnecessary css PR #1010:

Testing, Docs, and Builds

- Pip-install nbconvert on readthedocs.org PR #1069:
- Fix various doc build issues PR #1051:, PR #1050:, PR #1019:, and PR #1048:
- Add issue templates PR #1046:
- Added instructions for bumping the version forward when releasing PR #1034:
- Fix Testing on Windows PR #1030:
- Refactored `test_run_notebooks` PR #1015:
- Fixed documentation typos PR #1009:

16.4 5.5

The following 18 authors contributed 144 commits – Thank you all!

- Benjamin Ragan-Kelley
- Clayton A Davis
- DInne Bosman
- Doug Blank
- Henrique Silva
- Jeff Hale
- Lukasz Mitusinski
- M Pacer
- Maarten Breddels

- Madhumitha N
- Matthew Seal
- Paul Gowder
- Philipp A
- Rick Lupton
- Rüdiger Busche
- Thomas Kluyver
- Tyler Makaro
- WrRan

The full list of changes they made can be seen on GitHub

16.4.1 Significant Changes

Deprecations

Python 3.4 support was dropped. Many of our upstream libraries stopped supporting 3.4 and it was found that serious bugs were being caught during testing against those libraries updating past 3.4.

See PR #979 for details.

IPyWidget Support

Now when a notebook executing contains Jupyter Widgets, the state of all the widgets can be stored in the notebook's metadata. This allows rendering of the live widgets on, for instance nbviewer, or when converting to html.

You can tell nbconvert to not store the state using the `store_widget_state` argument:

```
jupyter nbconvert --ExecutePreprocessor.store_widget_state=False --to notebook --
↪execute mynotebook.ipynb
```

This widget rendering is not performed against a browser during execution, so only widget default states or states manipulated via user code will be calculated during execution. `%%javascript` cells will execute upon notebook rendering, enabling complex interactions to function as expected when viewed by a UI.

If you can't view widget results after execution, you may need to select *File → Trust Notebook* in the menu.

See PR #779, PR #900, and PR #983 for details.

Execute Preprocessor Rework

Based on monkey patching required in papermill the `run_cell` code path in the ExecutePreprocessor was reworked to allow for accessing individual message parses without reimplementing the entire function. Now there is a `process_message` function which take a ZeroMQ message and applies all of its side-effect updates on the cell/notebook objects before returning the output it generated, if it generated any such output.

The change required a much more extensive test suite covering cell execution as test coverage on the various, sometimes wonky, code paths made improvements and reworks impossible to prove undamaging. Now changes to kernel message processing has much better coverage, so future additions or changes with specs over time will be easier to add.

See PR #905 and PR #982 for details

Out Of Memory Kernel Failure Catches

When running out of memory on a machine, if the kernel process was killed by the operating system it would result in a timeout error at best and hang indefinitely at worst. Now regardless of timeout configuration, if the underlying kernel process dies before emitting any messages to the effect an exception will be raised notifying the consumer of the lost kernel within a few seconds.

See PR #959, PR #971, and PR #998 for details

Latex / PDF Template Improvements

The latex template was long overdue for improvements. The default template had a rewrite which makes exports for latex and pdf look a lot better. Code cells in particular render much better with line breaks and styling the more closely matches notebook browser rendering. Thanks t-makaro for the efforts here!

See PR #992 for details

16.4.2 Comprehensive notes

New Features

- IPyWidget Support PR #779, PR #900, and PR #983
- A new ClearMetadata Preprocessor is available PR #805:
- Support for pandoc 2 PR #964:
- New, and better, latex template PR #992:

Fixing Problems

- Refactored execute preprocessor to have a process_message function PR #905:
- Fixed OOM kernel failures hanging PR #959 and PR #971:
- Fixed latex export for svg data in python 3 PR #985:
- Enabled configuration to be shared to exporters from script exporter PR #993:
- Make latex errors less verbose PR #988:
- Typo in template syntax PR #984:
- Improved attachments +fix supporting non-unique names PR #980:
- PDFExporter “output_mimetype” traitlet is not longer ‘text/latex’ PR #972:
- FIX: respect wait for clear_output PR #969:
- address deprecation warning in cgi.escape PR #963:
- Correct inaccurate description of available LaTeX template PR #958:
- Fixed kernel death detection for executions with timeouts PR #998:
- Fixed export names for various templates PR #1000, PR #1001, and PR #1001:

Deprecations

- Dropped support for python 3.4 PR #979:
- Removed deprecated `export_by_name` PR #945:

Testing, Docs, and Builds

- Added tests for each branch in execute's run_cell method PR #982:
- Mention formats in –to options more clearly PR #991:
- Adds ascii output type to command line docs page, mention image folder output PR #956:
- Simplify setup.py PR #949:
- Use utf-8 encoding in execute_api example PR #921:
- Upgrade pytest on Travis PR #941:
- Fix LaTeX base template name in docs PR #940:
- Updated release instructions based on 5.4 release walk-through PR #887:
- Fixed broken link to jinja docs PR #997:

16.5 5.4.1

5.4.1 on Github

Thanks to the following 11 authors who contributed 57 commits.

- Benjamin Ragan-Kelley
- Carol Willing
- Clayton A Davis
- Daniel Rodriguez
- M Pacer
- Matthew Seal
- Matthias Geier
- Matthieu Parizy
- Rüdiger Busche
- Thomas Kluyver
- Tyler Makaro

16.5.1 Comprehensive notes

New Features

- Expose pygments styles PR #889:
- Tornado 6.0 support – Convert proxy handler from callback to coroutine PR #937:
- Add option to overwrite the highlight_code filter PR #877:

Fixing Problems

- Mathjax.tpl fix for rendering Latex in html PR #932:
- Backwards compatbility for empty kernel names PR #927 PR #924

Testing, Docs, and Builds

- DOC: Add missing language specification to code-block PR #882:

16.6 5.4

5.4 on Github

16.6.1 Significant Changes

Deprecations

Python 3.3 support was dropped. The version of python is no longer common and new versions have many fixes and interface improvements that warrant the change in support.

See PR #843 for implementation details.

Changes in how we handle metadata

There were a few new metadata fields which are now respected in nbconvert.

`nb.metadata.authors` metadata attribute will be respected in latex exports. Multiple authors will be added with `,` separation against their names.

`nb.metadata.title` will be respected ahead of `nb.metadata.name` for title assignment. This better matches with the notebook format.

`nb.metadata.filename` will override the default `output_filename_template` when extracting notebook resources in the *`ExtractOutputPreprocessor`*. The attribute is helpful for when you want to consistently fix to a particular output filename, especially when you need to set image filenames for your exports.

The `raises-exception` cell tag (`nb.cells[].metadata.tags[raises-exception]`) allows for cell exceptions to not halt execution. The tag is respected in the same way by nbval and other notebook interfaces. `nb.metadata.allow_errors` will apply this rule for all cells. This feature is toggleable with the `force_raise_errors` configuration option. Errors from executing the notebook can be allowed with a `raises-exception` tag on a single cell, or the `allow_errors` configurable option for all cells. An allowed

error will be recorded in notebook output, and execution will continue. If an error occurs when it is not explicitly allowed, a *CellExecutionError* will be raised. If `force_raise_errors` is True, *CellExecutionError* will be raised for any error that occurs while executing the notebook. This overrides both the `allow_errors` option and the `raises-exception` cell tags.

See PR #867, PR #703, PR #685, PR #672, and PR #684 for implementation changes.

Configurable kernel managers when executing notebooks

The kernel manager can now be optionally passed into the `ExecutePreprocessor.preprocess` and the `executenb` functions as the keyword argument `km`. This means that the kernel can be configured as desired before beginning preprocessing.

This is useful for executing in a context where the kernel has external dependencies that need to be set to non-default values. An example of this might be a Spark kernel where you wish to configure the Spark cluster location ahead of time without building a new kernel.

Overall the ExecutePreprocessor has been reworked to make it easier to use. Future releases will continue this trend to make this section of the code more inheritable and reusable by others. We encourage you read the source code for this version if you're interested in the detailed improvements.

See PR #852 for implementation changes.

Surfacing exporters in front-ends

Exporters are now exposed for front-ends to consume, including classic notebook. As an example, this means that latex exporter will be made available for latex 'text/latex' media type from the Download As interface.

See PR #759 and PR #864 for implementation changes.

Raw Templates

Template exporters can now be assigned raw templates as string attributes by setting the `raw_template` variable.

```
class AttrExporter(TemplateExporter):
    # If the class has a special template and you want it defined within the class
    raw_template = """{%- extends 'rst.tpl' -%}
{%- block in_prompt -%}
raw template
{%- endblock in_prompt -%}
    """
exporter_attr = AttrExporter()
output_attr, _ = exporter_attr.from_notebook_node(nb)
assert "raw template" in output_attr
```

See PR #675 for implementation changes.

New command line flags

The `--no-input` will hide input cells on export. This is great for notebooks which generate "reports" where you want the code that was executed to not appear by default in the extracts.

An alias for `notebook` was added to exporter commands. Now `--to ipynb` will behave as `--to notebook` does.

See PR #825 and PR #873 for implementation changes.

16.6.2 Comprehensive notes

New Features

- No input flag (`--no-input`) PR #825
- Add alias `--to ipynb` for notebook exporter PR #873
- Add `export_from_notebook` PR #864
- If set, use `nb.metadata.authors` for LaTeX author line PR #867
- Populate language_info metadata when executing PR #860
- Support for `\mathscr` PR #830
- Allow the execute preprocessor to make use of an existing kernel PR #852
- Refactor ExecutePreprocessor PR #816
- Update widgets CDN for ipywidgets 7 w/fallback PR #792
- Add support for adding custom exporters to the "Download as" menu. PR #759
- Enable ANSI underline and inverse PR #696
- Update notebook css to 5.4.0 PR #748
- Change default for slides to direct to the reveal cdn rather than locally PR #732
- Use "title" instead of "name" for metadata to match the notebook format PR #703
- Img filename metadata PR #685
- Added MathJax compatibility definitions PR #687
- Per cell exception PR #684
- Simple API for in-memory templates PR #674 PR #675
- Set BIBINPUTS and BSTINPUTS environment variables when making PDF PR #676
- If `nb.metadata.title` is set, default to that for notebook PR #672

Deprecations

- Drop support for python 3.3 PR #843

Fixing Problems

- Fix api break PR #872
- Don't remove empty cells by default PR #784
- Handle attached images in html converter PR #780
- No need to check for the channels already running PR #862
- Update `font-awesome` version for slides PR #793
- Properly treat JSON data PR #847
- Skip executing empty code cells PR #739
- Ppdate log.warn (deprecated) to log.warning PR #804
- Cleanup notebook.tex during PDF generation PR #768
- Windows unicode error fixed, nosetest added to setup.py PR #757
- Better content hiding; template & testing improvements PR #734
- Fix Jinja syntax in custom template example. PR #738
- Fix for an issue with empty math block PR #729
- Add parser for Multiline math for LaTeX blocks PR #716 PR #717
- Use defusedxml to parse potentially untrusted XML PR #708
- Fixes for traitlets 4.1 deprecation warnings PR #695

Testing, Docs, and Builds

- A couple of typos PR #870
- Add python_requires metadata. PR #871
- Document `--inplace` command line flag. PR #839
- Fix minor typo in `usage.rst` PR #863
- Add note about local `reveal_url_prefix` PR #844
- Move `onlyif_cmds_exist` decorator to test-specific utils PR #854
- Include LICENSE file in wheels PR #827
- Added Ubuntu Linux Instructions PR #724
- Check for too recent of pandoc version PR #814 PR #872
- Removing more nose remnants via dependencies. PR #758
- Remove offline statement and add some clarifications in slides docs PR #743
- Linkify PR number PR #710
- Added shebang for python PR #694

- Upgrade mistune dependency PR #705
- add feature to improve docs by having links to prs PR #662
- Update notebook CSS from version 4.3.0 to 5.1.0 PR #682
- Explicitly exclude or include all files in Manifest. PR #670

16.7 5.3.1

5.3.1 on Github

- MANIFEST.in updated to include `LICENSE` and `scripts/` when creating sdist. PR #666

16.8 5.3

5.3 on Github

16.8.1 Major features

Tag Based Element Filtering

For removing individual elements from notebooks, we need a way to signal to nbconvert that the elements should be removed. With this release, we introduce the use of tags for that purpose.

Tags are user-defined strings attached to cells or outputs. They are stored in cell or output metadata. For more on tags see the nbformat docs on cell metadata.

Usage:

1. Apply tags to the elements that you want to remove.

For removing an entire cell, the cell input, or all cell outputs apply the tag to the cell.

For removing individual outputs, put the tag in the output metadata using a call like `display(your_output_element, metadata={tags=[<your_tags_here>]})`.

NB: Use different tags depending on whether you want to remove the entire cell, the input, all outputs, or individual outputs.

2. Add the tags for removing the different kinds of elements to the following traitlets. Which kind of element you want to remove determines which traitlet you add the tags to.

The following traitlets remove elements of different kinds:

- `remove_cell_tags`: removes cells
- `remove_input_tags`: removes inputs
- `remove_all_outputs_tag`: removes all outputs
- `remove_single_output_tag`: removes individual outputs

16.8.2 Comprehensive notes

- new: configurable `browser` in ServePostProcessor PR #618
- new: `--clear-output` command line flag to clear output in-place PR #619
- new: remove elements based on tags with *`TagRemovePreprocessor`*. PR #640, PR #643
- new: CellExecutionError can now be imported from `nbconvert.preprocessors` PR #656
- new: slides now can enable scrolling and custom transitions PR #600
- docs: Release instructions for nbviewer-deploy
- docs: improved instructions for handling errors using the *`ExecutePreprocessor`* PR #656
- tests: better height/width metadata testing for images in rst & html PR #601 PR #602
- tests: normalise base64 output data to avoid false positives PR #650
- tests: normalise ipython traceback messages to handle old and new style PR #631
- bug: mathjax obeys `\\(\\)` & `\\[\\]` (both nbconvert & pandoc) PR #609 PR #617
- bug: specify default templates using extensions PR #639
- bug: fix pandoc version number PR #638
- bug: require recent mistune version PR #630
- bug: catch errors from IPython `execute_reply` and `error` messages PR #642
- nose completely removed & dependency dropped PR #595 PR #660
- mathjax processing in mistune now only uses inline grammar PR #611
- removeRegex now enabled by default on all TemplateExporters, does not remove cells with outputs PR #616
- validate notebook after applying each preprocessor (allowing additional attributes) PR #645
- changed COPYING.md to LICENSE for more standard licensing that GitHub knows how to read PR #654

16.9 5.2.1

5.2 on GitHub

16.9.1 Major features

In this release (along with the usual bugfixes and documentation improvements, which are legion) we have a few new major features that have been requested for a long time:

Global Content Filtering

You now have the ability to remove input or output from code cells, markdown cells and the input and output prompts. The easiest way to access all of these is by using traitlets like TemplateExporter.exclude_input = True (or, for example HTMLExporter.exclude_markdown = True if you wanted to make it specific to HTML output). On the command line if you just want to not have input or output prompts just use –no-prompt.

Execute notebooks from a function

You can now use the executenb function to execute notebooks as though you ran the execute preprocessor on the notebooks. It returns the standard notebook and resources options.

Remove cells based on regex pattern

This removes cells based on their matching a regex pattern (by default, empty cells). This is the RegexRemovePreprocessor.

Script exporter entrypoints for nonpython scripts

Now there is an entrypoint for having an exporter specific to the type of script that is being exported. While designed for use with the IRkernel in particular (with a script exporter focused on exporting R scripts) other non-python kernels that wish to have a language specific exporter can now surface that directly.

16.9.2 Comprehensive notes

- new: configurable ExecutePreprocessor.startup_timeout configurable PR #583
- new: RemoveCell preprocessor based on cell content (defaults to empty cell) PR #575
- new: function for executing notebooks: `executenb` PR #573
- new: global filtering to remove inputs, outputs, markdown cells (&c.), this works on all templates PR #554
- new: script exporter entrypoint PR #531
- new: configurable anchor link text (previously ¶) `HTMLExporter.anchor_link_text` PR #522
- new: configurable values for slides exporter PR #542 PR #558
- improved releases (how-to documentation, version-number generation and checking) PR #593
- doc improvements PR #593 PR #580 PR #565 PR #554
- language information from cell magics (for highlighting) is now included in more formats PR #586
- mathjax upgrades and cdn fixes PR #584 PR #567
- better CI PR #571 PR #540
- better traceback behaviour when execution errs PR #521
- deprecated nose test features removed PR #519
- bug fixed: we now respect width and height metadata on jpeg and png mimetype outputs PR #588
- bug fixed: now we respect the `resolve_references` filter in `report.tplx` PR #577
- bug fixed: output metadata now is removed by ClearOutputPreprocessor PR #569

- bug fixed: display id respected in execute preproessor PR #563
- bug fixed: dynamic defaults for optional jupyter_client import PR #559
- bug fixed: don't self-close non-void HTML tags PR #548
- buf fixed: upgrade jupyter_client dependency to 4.2 PR #539
- bug fixed: LaTeX output through md→LaTeX conversion shouldn't be touched PR #535
- bug fixed: now we escape < inside math formulas when converting to html PR #514

16.9.3 Credits

This release has been larger than previous releases. In it 33 authors contributed a total of 546 commits.

Many thanks to the following individuals who contributed to this release (in alphabetical order):

- Adam Chainz
- Andreas Mueller
- Bartosz T
- Benjamin Ragan-Kelley
- Carol Willing
- Damián Avila
- Elliot Marsden
- Gao, Xiang
- Jaeho Shin
- Jan Schulz
- Jeremy Kun
- Jessica B. Hamrick
- John B Nelson
- juhasch
- Livia Barazzetti
- M Pacer
- Matej Urbas
- Matthias Bussonnier
- Matthias Geier
- Maximilian Albert
- Michael Scott Cuthbert
- Nicholas Bollweg
- Paul Gowder
- Paulo Villegas
- Peter Parente
- Philipp A

- Scott Sanderson
- Srinivas Reddy Thatiparthy
- Sylvain Corlay
- Thomas Kluyver
- Till Hoffmann
- Xiang Gao
- YuviPanda

16.10 5.1.1

5.1.1 on GitHub

- fix version numbering because of incomplete previous version number

16.11 5.1

5.1 on GitHub

- improved CSS (specifically tables, in line with notebook) PR #498
- improve in-memory templates handling PR #491
- test improvements PR #516 PR #509 PR #505
- new configuration option: IOPub timeout PR #513
- doc improvements PR #489 PR #500 PR #493 PR #506
- newly customizable: output prompt PR #500
- more python2/3 compatibile unicode handling PR #502

16.12 5.0

5.0 on GitHub

- Use **`xelatex`** by default for latex export, improving unicode and font support.
- Use entrypoints internally to access Exporters, allowing for packages to declare custom exporters more easily.
- New ASCIIDoc Exporter.
- New preprocessor for sanitised html output.
- New general `convert_pandoc` filter to reduce the need to hard-code lists of filters in templates.
- Use pytest, nose dependency to be removed.
- Refactored Exporter code to avoid ambiguity and cyclic dependencies.
- Update to traitlets 4.2 API.
- Fixes for Unicode errors when showing execution errors on Python 2.
- Default math font matches default Palatino body text font.

- General documentation improvements. For example, testing, installation, custom exporters.
- Improved link handling for LaTeX output
- Refactored the automatic id generation.
- New kernel_manager_class configuration option for allowing systems to be set up to resolve kernels in different ways.
- Kernel errors now will be logged for debugging purposes when executing notebooks.

16.13 4.3

4.3 on GitHub

- added live widget rendering for html output, nbviewer by extension

16.14 4.2

4.2 on GitHub

- *Custom Exporters* can be provided by external packages, and registered with nbconvert via setuptools entry-points.
- allow nbconvert reading from stdin with `--stdin` option (write into `notebook` basename)
- Various ANSI-escape fixes and improvements
- Various LaTeX/PDF export fixes
- Various fixes and improvements for executing notebooks with `--execute`.

16.15 4.1

4.1 on GitHub

- setuptools fixes for entrypoints on Windows
- various fixes for exporters, including slides, latex, and PDF
- fixes for exceptions met during execution
- include markdown outputs in markdown/html exports

16.16 4.0

4.0 on GitHub

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

nbconvert, 77
nbconvert.exporters, 78
nbconvert.filters, 86
nbconvert.nbconvertapp, 77
nbconvert.postprocessors, 89
nbconvert.writers, 89

INDEX

Symbols

__init__() (*nbconvert.exporters.Exporter method*), 79
__init__() (*nbconvert.exporters.TemplateExporter method*), 81
__init__() (*nbconvert.preprocessors.Preprocessor method*), 83
__init__() (*nbconvert.writers.WriterBase method*), 89

A

add_anchor() (*in module nbconvert.filters*), 86
add_prompts() (*in module nbconvert.filters*), 86
ansi2html() (*in module nbconvert.filters*), 86
ansi2latex() (*in module nbconvert.filters*), 86
ascii_only() (*in module nbconvert.filters*), 86

C

CellExecutionError (*class in nbconvert.preprocessors*), 85
citation2latex() (*in module nbconvert.filters*), 86
ClearMetadataPreprocessor (*class in nbconvert.preprocessors*), 84
ClearOutputPreprocessor (*class in nbconvert.preprocessors*), 84
coalesce_streams() (*in module nbconvert.preprocessors*), 85
comment_lines() (*in module nbconvert.filters*), 86
convert_notebooks() (*nbconvert.nbconvertapp.NbConvertApp method*), 77
convert_pandoc() (*in module nbconvert.filters*), 87
convert_single_notebook() (*nbconvert.nbconvertapp.NbConvertApp method*), 77
ConvertFiguresPreprocessor (*class in nbconvert.preprocessors*), 84
CSSHTMLHeaderPreprocessor (*class in nbconvert.preprocessors*), 84

D

DataTypeFilter (*class in nbconvert.filters*), 86
DebugWriter (*class in nbconvert.writers*), 89

E

escape_latex() (*in module nbconvert.filters*), 86
ExecutePreprocessor (*class in nbconvert.preprocessors*), 85
export() (*in module nbconvert.exporters*), 78
export_single_notebook() (*nbconvert.nbconvertapp.NbConvertApp method*), 78
Exporter (*class in nbconvert.exporters*), 79
ExtractOutputPreprocessor (*class in nbconvert.preprocessors*), 84

F

FilesWriter (*class in nbconvert.writers*), 89
from_file() (*nbconvert.exporters.Exporter method*), 80
from_file() (*nbconvert.exporters.TemplateExporter method*), 81
from_filename() (*nbconvert.exporters.Exporter method*), 79
from_filename() (*nbconvert.exporters.TemplateExporter method*), 81
from_notebook_node() (*nbconvert.exporters.Exporter method*), 79
from_notebook_node() (*nbconvert.exporters.TemplateExporter method*), 81

G

get_export_names() (*in module nbconvert.exporters*), 79
get_exporter() (*in module nbconvert.exporters*), 79
get_lines() (*in module nbconvert.filters*), 86

H

Highlight2HTML (*class in nbconvert.filters*), 87
Highlight2Latex (*class in nbconvert.filters*), 87
HighlightMagicsPreprocessor (*class in nbconvert.preprocessors*), 84

html2text() (*in module nbconvert.filters*), 87
HTMLExporter (*class in nbconvert.exporters*), 82

I

indent() (*in module nbconvert.filters*), 87
init_notebooks() (*nbconvert.nbconvertapp.NbConvertApp method*), 77
init_single_notebook_resources() (*nbconvert.nbconvertapp.NbConvertApp method*), 77
ipython2python() (*in module nbconvert.filters*), 87

L

LatexExporter (*class in nbconvert.exporters*), 82
LatexPreprocessor (*class in nbconvert.preprocessors*), 84

M

markdown2html() (*in module nbconvert.filters*), 87
markdown2latex() (*in module nbconvert.filters*), 87
markdown2rst() (*in module nbconvert.filters*), 88
MarkdownExporter (*class in nbconvert.exporters*), 82
module
    nbconvert, 77
    nbconvert.exporters, 78
    nbconvert.filters, 86
    nbconvert.nbconvertapp, 77
    nbconvert.postprocessors, 89
    nbconvert.writers, 89

N

nbconvert
    module, 77
nbconvert.exporters
    module, 78
nbconvert.filters
    module, 86
nbconvert.nbconvertapp
    module, 77
nbconvert.postprocessors
    module, 89
nbconvert.writers
    module, 89
NbConvertApp (*class in nbconvert.nbconvertapp*), 77
NotebookExporter (*class in nbconvert.exporters*), 82

P

path2url() (*in module nbconvert.filters*), 88
PDFExporter (*class in nbconvert.exporters*), 82
posix_path() (*in module nbconvert.filters*), 88
postprocess() (*nbconvert.postprocessors.PostProcessorBase method*), 89
postprocess() (*nbconvert.postprocessors.ServePostProcessor method*), 90
postprocess_single_notebook() (*nbconvert.nbconvertapp.NbConvertApp method*), 78
PostProcessorBase (*class in nbconvert.postprocessors*), 89
preprocess() (*nbconvert.preprocessors.ExecutePreprocessor method*), 85
preprocess() (*nbconvert.preprocessors.Preprocessor method*), 83
preprocess_cell() (*nbconvert.preprocessors.Preprocessor method*), 83
Preprocessor (*class in nbconvert.preprocessors*), 83
prevent_list_blocks() (*in module nbconvert.filters*), 88
PythonExporter (*class in nbconvert.exporters*), 83

R

RegexRemovePreprocessor (*class in nbconvert.preprocessors*), 84
register_filter() (*nbconvert.exporters.TemplateExporter method*), 82
register_preprocessor() (*nbconvert.exporters.Exporter method*), 80
register_preprocessor() (*nbconvert.exporters.TemplateExporter method*), 82
RSTExporter (*class in nbconvert.exporters*), 83

S

ServePostProcessor (*class in nbconvert.postprocessors*), 90
SlidesExporter (*class in nbconvert.exporters*), 82
StdoutWriter (*class in nbconvert.writers*), 89
strip_ansi() (*in module nbconvert.filters*), 88
strip_dollars() (*in module nbconvert.filters*), 88
strip_files_prefix() (*in module nbconvert.filters*), 88
SVG2PDFPreprocessor (*class in nbconvert.preprocessors*), 84

T

TagRemovePreprocessor (*class in nbconvert.preprocessors*), 84

TemplateExporter (*class in nbconvert.exporters*), 80

W

WebPDFExporter (*class in nbconvert.exporters*), 82
wrap_text() (*in module nbconvert.filters*), 88
write() (*nbconvert.writers.WriterBase method*), 89
write_single_notebook() (*nbconvert.nbconvertapp.NbConvertApp method*), 78
WriterBase (*class in nbconvert.writers*), 89