
nbconvert Documentation

Release 4.1.0

Jupyter Development Team

Mar 09, 2017

CONTENTS

- 1 Installing nbconvert 3**
 - 1.1 Pandoc 3
- 2 Command-Line Usage 5**
- 3 LaTeX citations 9**
- 4 Python API for working with nbconvert 11**
 - 4.1 NbConvertApp 11
 - 4.2 Exporters 12
 - 4.3 Preprocessors 20
 - 4.4 Filters 21
 - 4.5 Writers 24
 - 4.6 Postprocessors 24
- 5 Changes in nbconvert 27**
 - 5.1 4.1 27
 - 5.2 4.0 27
- 6 Indices and tables 29**
- Python Module Index 31**

The `nbconvert` tool allows you to convert an `.ipynb` notebook document file into various static formats.

Currently, `nbconvert` is provided as a command line tool, run as a script using Jupyter. It also powers the ‘Download as’ feature within the Jupyter Notebook web app.

Contents:

INSTALLING NBCONVERT

See also:

Installing Jupyter Nbconvert is part of the Jupyter ecosystem.

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

```
pip install nbconvert
# OR
conda install nbconvert
```

If you're new to Python, we recommend installing [Anaconda](#), a Python distribution which includes nbconvert and the other Jupyter components.

1.1 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](#).

COMMAND-LINE USAGE

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 given by the `FORMAT` string.

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

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

Jupyter provides a few templates for some output formats, and these can be specified via an additional `--template` argument.

The currently supported export formats are:

- `--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.
- `--to latex`
Latex export. This generates `NOTEBOOK_NAME.tex` file, ready for export.
 - `--template article` (default)
Latex article, derived from Sphinx's howto template.
 - `--template report`
Latex report, providing a table of contents and chapters.
 - `--template basic`
Very basic latex output - mainly meant as a starting point for custom templates.
- `--to pdf`
Generates a PDF via latex. Supports the same templates as `--to latex`.
- `--to slides`
This generates a Reveal.js HTML slideshow. It must be served by an HTTP server. The easiest way to do this is adding `--post serve` on the command-line. The `serve` post-processor proxies Reveal.js requests to a CDN if no local Reveal.js library is present. To make slides that don't require an internet connection, just place

the Reveal.js library in the same directory where your `your_talk.slides.html` is located, or point to another directory using the `--reveal-prefix` alias.

- `--to markdown`

Simple markdown output. Markdown cells are unaffected, and code cells indented 4 spaces.

- `--to rst`

Basic reStructuredText output. Useful as a starting point for embedding notebooks in Sphinx docs.

- `--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 an Jupyter session.

- `--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
```

will open the notebook, execute it, capture new output, and save the result in `mynotebook.nbconvert.ipynb`. 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.

```
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
```

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
```


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.

PYTHON API FOR WORKING WITH NBCONVERT

Contents:

4.1 NbConvertApp

See also:

`/config_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 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*)

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

init_single_notebook_resources (*notebook_filename*)

Step 1: Initialize resources

This initializes the resources dictionary for a single notebook. This method should return the resources dictionary, and 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

export_single_notebook (*notebook_filename, resources*)

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.

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.

postprocess_single_notebook (*write_results*)

Step 4: Postprocess the notebook

This postprocesses the notebook after it has been written, taking as an argument the results of writing the notebook to file. This only actually does anything if a postprocessor has actually been specified.

4.2 Exporters

See also:

/config_options Configurable options for the nbconvert application

class nbconvert.exporters.**Exporter** (*config=None, **kw*)

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** (*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** (*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, resources=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** (*config=None, **kw*)

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.

- `add_anchor`
- `add_prompts`
- `ansi2html`
- `ansi2latex`
- `ascii_only`
- `citation2latex`
- `comment_lines`
- `escape_latex`
- `filter_data_type`
- `get_lines`
- `get_metadata`
- `highlight2html`
- `highlight2latex`
- `html2text`
- `indent`
- `ipython2python`
- `markdown2html`
- `markdown2latex`
- `markdown2rst`
- `path2url`
- `posix_path`
- `prevent_list_blocks`
- `strip_ansi`
- `strip_dollars`
- `strip_files_prefix`

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

from_filename (*filename, resources=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*) –

4.2.1 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` (*config=None*, ***kw*)
Exports to an IPython notebook.

class `nbconvert.exporters.HTMLExporter` (*config=None*, ***kw*)
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` (*config=None*, ***kw*)
Exports HTML slides with reveal.js

class `nbconvert.exporters.LatexExporter` (*config=None*, ***kw*)
Exports to a Latex template. Inherit from this class if your template is LaTeX based and you need custom transformers/filters. Inherit from it if you are writing your own HTML template and 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` (*config=None*, ***kw*)
Exports to a markdown document (.md)

class `nbconvert.exporters.PDFExporter` (*config=None*, ***kw*)
Writer designed to write to PDF files

class `nbconvert.exporters.PythonExporter` (*config=None*, ***kw*)
Exports a Python code file.

class `nbconvert.exporters.RSTExporter` (*config=None*, ***kw*)
Exports restructured text documents.

4.2.2 Specialized exporter functions

These functions are essentially convenience functions that wrap the functionality of the classes documented in the previous section.

`nbconvert.exporters.export_custom` (**args*, ***kwargs*)
Export a notebook object to custom format

nb [`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`] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [`Exporter`] Instance of the `Exporter` class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

`nbconvert.exporters.export_html(*args, **kwargs)`

Export a notebook object to html format

nb [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] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [Exporter] Instance of the Exporter class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

`nbconvert.exporters.export_slides(*args, **kwargs)`

Export a notebook object to slides format

nb [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] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [Exporter] Instance of the Exporter class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

`nbconvert.exporters.export_latex(*args, **kwargs)`

Export a notebook object to latex format

nb [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] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [Exporter] Instance of the Exporter class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

`nbconvert.exporters.export_pdf(*args, **kwargs)`

Export a notebook object to pdf format

nb [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] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [Exporter] Instance of the Exporter class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

`nbconvert.exporters.export_markdown(*args, **kwargs)`

Export a notebook object to markdown format

nb [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] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [Exporter] Instance of the Exporter class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

`nbconvert.exporters.export_python(*args, **kwargs)`

Export a notebook object to python format

nb [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] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [Exporter] Instance of the Exporter class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

`nbconvert.exporters.export_rst(*args, **kwargs)`

Export a notebook object to rst format

nb [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] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [Exporter] Instance of the Exporter class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

`nbconvert.exporters.export_script(*args, **kwargs)`

Export a notebook object to script format

nb [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] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [Exporter] Instance of the Exporter class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

`nbconvert.exporters.export_by_name(*args, **kwargs)`

Export a notebook object to a template type by its name. Reflection (Inspect) is used to find the template's corresponding explicit export method defined in this module. That method is then called directly.

Parameters **format_name** (*str*) – Name of the template style to export to.

nb [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] Jinja 2 output. This is the resulting converted notebook.

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

exporter_instance [Exporter] Instance of the Exporter class used to export the document. Useful to caller because it provides a 'file_extension' property which specifies what extension the output should be saved as.

Return type tuple

Notes

WARNING: API WILL CHANGE IN FUTURE RELEASES OF NBCONVERT

4.3 Preprocessors

See also:

`/config_options` Configurable options for the nbconvert application

class `nbconvert.preprocessors.Preprocessor` (***kw*)

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 informations.

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

4.3.1 Specialized preprocessors

class `nbconvert.preprocessors.ConvertFiguresPreprocessor` (***kw*)

Converts all of the outputs in a notebook from one format to another.

class `nbconvert.preprocessors.SVG2PDFPreprocessor` (***kw*)

Converts all of the outputs in a notebook from SVG to PDF.


```
class nbconvert.preprocessors.ExtractOutputPreprocessor (**kw)
    Extracts all of the outputs from the notebook file. The extracted outputs are returned in the 'resources' dictionary.

class nbconvert.preprocessors.LatexPreprocessor (**kw)
    Preprocessor for latex destined documents.

    Mainly populates the latex key in the resources dict, adding definitions for pygments highlight styles.

class nbconvert.preprocessors.CSSHTMLHeaderPreprocessor (*pargs, **kwargs)
    Preprocessor used to pre-process notebook for HTML output. Adds IPython notebook front-end CSS and Pygments CSS to HTML output.

class nbconvert.preprocessors.HighlightMagicsPreprocessor (config=None, **kw)
    Detects and tags code cells that use a different languages than Python.

class nbconvert.preprocessors.ClearOutputPreprocessor (**kw)
    Removes the output from all code cells in a notebook.

class nbconvert.preprocessors.ExecutePreprocessor (**kw)
    Executes all the cells in a notebook

nbconvert.preprocessors.coalesce_streams (nb, resources)
    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

4.4 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)
    Add 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** (*str*) – Text containing ansi colors to convert to html

```
nbconvert.filters.ansi2latex (text)
    Converts ansi formatted text to latex version

    based on https://bitbucket.org/birkenfeld/sphinx-contrib/ansi.py

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 tranformation 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(**kw)`

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.

class `nbconvert.filters.Highlight2HTML(pygments_lexer=None, **kwargs)`

class `nbconvert.filters.Highlight2Latex(pygments_lexer=None, **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 <http://johnmacfarlane.net/pandoc/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.

4.5 Writers

See also:

`/config_options` Configurable options for the nbconvert application

class `nbconvert.writers.WriterBase` (*config=None*, ***kw*)

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.

4.5.1 Specialized writers

class `nbconvert.writers.DebugWriter` (*config=None*, ***kw*)

Consumes output from `nbconvert.export...()` methods and writes usefull 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` (***kw*)

Consumes nbconvert output and produces files.

class `nbconvert.writers.StdoutWriter` (*config=None*, ***kw*)

Consumes output from `nbconvert.export...()` methods and writes to the stdout stream.

4.6 Postprocessors

See also:

`/config_options` Configurable options for the nbconvert application

class `nbconvert.postprocessors.PostProcessorBase` (***kw*)

postprocess (*input*)

Post-process output from a writer.

4.6.1 Specialized postprocessors

class nbconvert.postprocessors.**ServePostProcessor** (**kw)

Post processor designed to serve files

Proxies reveal.js requests to a CDN if no local reveal.js is present

CHANGES IN NBCONVERT

5.1 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

5.2 4.0

[4.0 on GitHub](#)

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

n

- `nbconvert`, [11](#)
- `nbconvert.exporters`, [12](#)
- `nbconvert.filters`, [21](#)
- `nbconvert.nbconvertapp`, [11](#)
- `nbconvert.postprocessors`, [24](#)
- `nbconvert.preprocessors`, [20](#)
- `nbconvert.writers`, [24](#)

Symbols

`__init__()` (nbconvert.exporters.Exporter method), 12
`__init__()` (nbconvert.exporters.TemplateExporter method), 14
`__init__()` (nbconvert.preprocessors.Preprocessor method), 20
`__init__()` (nbconvert.writers.WriterBase method), 24

A

`add_anchor()` (in module nbconvert.filters), 21
`add_prompts()` (in module nbconvert.filters), 21
`ansi2html()` (in module nbconvert.filters), 21
`ansi2latex()` (in module nbconvert.filters), 21
`ascii_only()` (in module nbconvert.filters), 21

C

`citation2latex()` (in module nbconvert.filters), 21
`ClearOutputPreprocessor` (class in nbconvert.preprocessors), 21
`coalesce_streams()` (in module nbconvert.preprocessors), 21
`comment_lines()` (in module nbconvert.filters), 22
`convert_notebooks()` (nbconvert.nbconvertapp.NbConvertApp method), 11
`convert_single_notebook()` (nbconvert.nbconvertapp.NbConvertApp method), 11
`ConvertFiguresPreprocessor` (class in nbconvert.preprocessors), 20
`CSSHTMLHeaderPreprocessor` (class in nbconvert.preprocessors), 21

D

`DataTypeFilter` (class in nbconvert.filters), 22
`DebugWriter` (class in nbconvert.writers), 24

E

`escape_latex()` (in module nbconvert.filters), 22
`ExecutePreprocessor` (class in nbconvert.preprocessors), 21

`export_by_name()` (in module nbconvert.exporters), 19
`export_custom()` (in module nbconvert.exporters), 15
`export_html()` (in module nbconvert.exporters), 16
`export_latex()` (in module nbconvert.exporters), 16
`export_markdown()` (in module nbconvert.exporters), 17
`export_pdf()` (in module nbconvert.exporters), 17
`export_python()` (in module nbconvert.exporters), 18
`export_rst()` (in module nbconvert.exporters), 18
`export_script()` (in module nbconvert.exporters), 19
`export_single_notebook()` (nbconvert.nbconvertapp.NbConvertApp method), 11
`export_slides()` (in module nbconvert.exporters), 16
`Exporter` (class in nbconvert.exporters), 12
`ExtractOutputPreprocessor` (class in nbconvert.preprocessors), 20

F

`FilesWriter` (class in nbconvert.writers), 24
`from_file()` (nbconvert.exporters.Exporter method), 12
`from_file()` (nbconvert.exporters.TemplateExporter method), 14
`from_filename()` (nbconvert.exporters.Exporter method), 12
`from_filename()` (nbconvert.exporters.TemplateExporter method), 14
`from_notebook_node()` (nbconvert.exporters.Exporter method), 12
`from_notebook_node()` (nbconvert.exporters.TemplateExporter method), 14

G

`get_lines()` (in module nbconvert.filters), 22

H

`Highlight2HTML` (class in nbconvert.filters), 22
`Highlight2Latex` (class in nbconvert.filters), 22
`HighlightMagicsPreprocessor` (class in nbconvert.preprocessors), 21
`html2text()` (in module nbconvert.filters), 22
`HTMLExporter` (class in nbconvert.exporters), 15

I

indent() (in module nbconvert.filters), 22
 init_notebooks() (nbconvert.nbconvertapp.NbConvertApp method), 11
 init_single_notebook_resources() (nbconvert.nbconvertapp.NbConvertApp method), 11
 ipython2python() (in module nbconvert.filters), 23

L

LatexExporter (class in nbconvert.exporters), 15
 LatexPreprocessor (class in nbconvert.preprocessors), 21

M

markdown2html() (in module nbconvert.filters), 23
 markdown2latex() (in module nbconvert.filters), 23
 markdown2rst() (in module nbconvert.filters), 23
 MarkdownExporter (class in nbconvert.exporters), 15

N

nbconvert (module), 11
 nbconvert.exporters (module), 12
 nbconvert.filters (module), 21
 nbconvert.nbconvertapp (module), 11
 nbconvert.postprocessors (module), 24
 nbconvert.preprocessors (module), 20
 nbconvert.writers (module), 24
 NbConvertApp (class in nbconvert.nbconvertapp), 11
 NotebookExporter (class in nbconvert.exporters), 15

P

path2url() (in module nbconvert.filters), 23
 PDFExporter (class in nbconvert.exporters), 15
 posix_path() (in module nbconvert.filters), 23
 postprocess() (nbconvert.postprocessors.PostProcessorBase method), 24
 postprocess_single_notebook() (nbconvert.nbconvertapp.NbConvertApp method), 12
 PostProcessorBase (class in nbconvert.postprocessors), 24
 preprocess() (nbconvert.preprocessors.Preprocessor method), 20
 preprocess_cell() (nbconvert.preprocessors.Preprocessor method), 20
 Preprocessor (class in nbconvert.preprocessors), 20
 prevent_list_blocks() (in module nbconvert.filters), 23
 PythonExporter (class in nbconvert.exporters), 15

R

register_filter() (nbconvert.exporters.TemplateExporter method), 14

register_preprocessor() (nbconvert.exporters.Exporter method), 13
 register_preprocessor() (nbconvert.exporters.TemplateExporter method), 14

RSTExporter (class in nbconvert.exporters), 15

S

ServePostProcessor (class in nbconvert.postprocessors), 25
 SlidesExporter (class in nbconvert.exporters), 15
 StdoutWriter (class in nbconvert.writers), 24
 strip_ansi() (in module nbconvert.filters), 23
 strip_dollars() (in module nbconvert.filters), 23
 strip_files_prefix() (in module nbconvert.filters), 23
 SVG2PDFPreprocessor (class in nbconvert.preprocessors), 20

T

TemplateExporter (class in nbconvert.exporters), 13

W

wrap_text() (in module nbconvert.filters), 23
 write() (nbconvert.writers.WriterBase method), 24
 write_single_notebook() (nbconvert.nbconvertapp.NbConvertApp method), 11
 WriterBase (class in nbconvert.writers), 24