Jupyter Format

Release 67bf141

Matthias Geier

2023-09-02

Contents

The following section was generated from `doc/index.jupyter`

Source code repository and issue tracker: https://github.com/mgeier/jupyter-format/

Note

All source files for this documentation are Jupyter notebooks stored in the proposed new storage format.

WARNING

This is in an experimental state. Neither the format nor the implementation is stable and either can change at any time without prior notice.

The following section was generated from `doc/installation.jupyter`

1 Installation

Since `jupyter_format` is in an experimental state, it is deliberately *not* available on PyPI[1].

But if you want to play around with it, you can get it from Github[2] and make an "editable" installation with

```
git clone https://github.com/mgeier/jupyter-format.git
cd jupyter-format
python3 -m pip install -e .
```

Depending on your Python installation, you may have to use `python` instead of `python3`.

If you don't need a local Git checkout, you can also directly install it with

```
python3 -m pip install git+https://github.com/mgeier/jupyter-format.git@master
```

1.1 JupyterLab and Classic Notebook Integration

If you want to be able to load and save Jupyter notebooks in the new format, you can specify a custom "contents manager" in your configuration.

You can find your configuration directory with

```
python3 -m jupyter --config-dir
```

If you don't yet have a configuration file there, you can generate it with JupyterLab:

```
python3 -m jupyterlab --generate-config
```

If you still use the Classic Notebook, generate the file with

```
python3 -m notebook --generate-config
```

To enable the custom "contents manager", simply open the file `jupyter_notebook_config.py`and add this line:

```
c.NotebookApp.contents_manager_class = 'jupyter_format.contents_manager.
↪FileContentsManager'
```

At the time of writing this, JupyterLab is not yet able to open `*.jupyter` files by double-clicking. You have to right-click and use "Open With" → "Notebook". See also https://github.com/jupyterlab/jupyterlab/issues/4924 and https://github.com/jupyterlab/jupyterlab/pull/5247.

[1] https://pypi.org
[2] https://github.com/mgeier/jupyter-format

1.2 Usage on Binder

If you want to use the new format on https://mybinder.org (which is great!), just create a configuration file in your repository using the path

```
.jupyter/jupyter_notebook_config.py
```

... and put the above-mentioned line in it.

doc/installation.jupyter ends here.

The following section was generated from doc/motivation.jupyter

2 Motivation

2.1 Status Quo

The original format for Jupyter[3] notebooks uses JSON[4] as underlying storage format. This has the great advantage that such files are very easy to handle programmatically in many different environments, because JSON parsers are readily available for many programming languages.

One disadvantage, however, is that the format is only semi-human-readable and not very well human-editable. All textual content (e.g. text in Markdown cells and source code in code cells) is stored in lists of JSON strings – one string for each line. This means that each line is surrounded by quotes (`"`) and strings are separated by commas (`,`), while lists of strings are surrounded by brackets (`[` and `]`). On top of that, several common characters are not allowed in JSON strings, which means that they have to be escaped by backslashes, e.g. `\"` and `\n`. And since a backslash is used for escaping, a literal backslash occurring in the text (which is quite common in programming languages and markup languages) has to be escaped itself (`\\`).

As an example, let's create a notebook containing the previous two sentences:

[1]:
```
import nbformat
nb = nbformat.v4.new_notebook()
nb.cells.append(nbformat.v4.new_markdown_cell(
r"""On top of that,
several common characters are not allowed in JSON strings,
which means that they have to be escaped by backslashes,
e.g. `\"` and `\n`.
And since a backslash is used for escaping,
a literal backslash occurring in the text
(which is quite common in programming languages and markup languages)
has to be escaped itself (`\\`)."""))
```

The JSON-based storage of this minimal notebook looks like this:

[2]:
```
print(nbformat.writes(nb))
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "c443c1a6",
   "metadata": {},
   "source": [
```

(continues on next page)

[3] https://jupyter.org/
[4] http://json.org/

(continued from previous page)

```
    "On top of that,\n",
    "several common characters are not allowed in JSON strings,\n",
    "which means that they have to be escaped by backslashes,\n",
    "e.g. `\\\"` and `\\n`.\n",
    "And since a backslash is used for escaping,\n",
    "a literal backslash occurring in the text\n",
    "(which is quite common in programming languages and markup languages)\n",
    "has to be escaped itself (`\\\\`)."
   ]
  }
 ],
 "metadata": {},
 "nbformat": 4,
 "nbformat_minor": 5
}
```

Escaped characters and JSON syntax elements make this harder than necessary to read, and even harder to modify with a text editor. When editing this by hand, it is easy to mess up the JSON representation by e.g. forgetting a comma.

As a comparison, the same notebook is stored like this in the proposed new format:

[3]:
```
import jupyter_format
print(jupyter_format.serialize(nb))
```

```
nbformat 4
nbformat_minor 5
markdown
    On top of that,
    several common characters are not allowed in JSON strings,
    which means that they have to be escaped by backslashes,
    e.g. `\"` and `\n`.
    And since a backslash is used for escaping,
    a literal backslash occurring in the text
    (which is quite common in programming languages and markup languages)
    has to be escaped itself (`\\`).
```

This is exactly the same as the original Markdown content, except that it is indented by 4 spaces.

2.2 YAML as Almost-Solution

It has been known for a long time (probably since the inception of Jupyter/IPython notebooks) that lists of JSON strings are not nicely readable for humans. An obvious alternative would be to use YAML[5], which provides multiple ways to store text content. One of those ways is the so-called literal style[6], which doesn't require any escaping, making the text much more readable.

This was already suggested in several blog posts:

- https://matthiasbussonnier.com/posts/05-YAML%20Notebook.html
- http://droettboom.com/blog/2018/01/18/diffable-jupyter-notebooks/

And there are even some implementations available:

[5] https://yaml.org
[6] https://yaml.org/spec/1.2/spec.html#style/block/literal

- https://github.com/prabhuramachandran/ipyaml
- https://github.com/mdboom/nbconvert_vc

This is an example how a YAML-based storage format could look like:

[4]:
```
yaml_content = """
nbformat: 4
nbformat_minor: 2
cells:
- cell_type: markdown
  source: |+2
    # A Jupyter Notebook

    This is a code cell:
  metadata: {}
- cell_type: code
  source: |+2
    print('Hello, world!')
  outputs:
  - output_type: stream
    name: stdout
    text: |+2
      Hello, world!
  execution_count: 1
  metadata: {}
metadata: {}
"""
```

This is valid YAML, compatible with both version 1.1 and 1.2.

Let's use PyYAML[7] to read this:

[5]:
```
import yaml
nb_dict = yaml.safe_load(yaml_content)
nb_dict
```

[5]:
```
{'nbformat': 4,
 'nbformat_minor': 2,
 'cells': [{'cell_type': 'markdown',
   'source': '# A Jupyter Notebook\n\nThis is a code cell:\n',
   'metadata': {}},
  {'cell_type': 'code',
   'source': "print('Hello, world!')\n",
   'outputs': [{'output_type': 'stream',
     'name': 'stdout',
     'text': 'Hello, world!\n'}],
   'execution_count': 1,
   'metadata': {}}],
 'metadata': {}}
```

This Python dictionary can easily be converted to a notebook node:

[6]:
```
nb = nbformat.from_dict(nb_dict)
```

And we can use `nbconvert` to convert this to HTML:

[7] https://pyyaml.org

[7]:
```
from nbconvert.exporters import HTMLExporter
html_content, resources = HTMLExporter().from_notebook_node(nb)
```

[8]:
```
import urllib
data_uri = 'data:text/html;charset=utf-8,' + urllib.parse.quote(html_content)
```

[9]:
```
from IPython.display import IFrame
IFrame(data_uri, width='100%', height='250')
```

[9]:
```
<IPython.lib.display.IFrame at 0x7f903247da50>
```

This looks promising, doesn't it?

The problem is, as so often, in the details. It's great that we can use *literal style* without littering the text with quotes and escape characters, but sadly, YAML only allows printable characters[8]. This means that we cannot use some control characters which might occur in cell outputs, for example ANSI escape characters.

There are two options here:

- Go back to escaped strings, at least in some circumstances. But that's exactly what we wanted to avoid by using YAML!
- Don't use YAML after all

2.3 Other Partial Solutions

Some alternative notebook formats are supported by the very popular projects https://github.com/aaren/notedown (Markdown) and https://github.com/mwouts/jupytext (Markdown, Rmd, Julia/Python/R-scripts etc.).

Those can be very useful, but none of them can store cell outputs, therefore they cannot be a full replacement for the current storage format.

2.4 The Need for a Custom Format

Looks like none of the existing formats are sufficient. Probably we can achieve our goals with a custom format.

Having to implement a custom parser for such a custom format is of course a disadvantage, but if we keep it really simple, probably we can get away with it?

Remember the YAML example from *above* (page 4)?

[10]:
```
print(yaml_content)
```

```
nbformat: 4
nbformat_minor: 2
cells:
- cell_type: markdown
  source: |+2
    # A Jupyter Notebook

    This is a code cell:
```

(continues on next page)

[8] https://yaml.org/spec/1.2/spec.html#printable%20character

(continued from previous page)

```
  metadata: {}
- cell_type: code
  source: |+2
    print('Hello, world!')
  outputs:
  - output_type: stream
    name: stdout
    text: |+2
      Hello, world!
  execution_count: 1
  metadata: {}
metadata: {}
```

The contained text (Markdown and Python source code) is quite readable, but it is still stuffed with many distracting things inbetween.

Since we are not limited by YAML anymore, we can agressively reduce this to only contain the absolutely necessary information:

[11]:
```
content = """nbformat 4
nbformat_minor 2
markdown
    # A Jupyter Notebook

    This is a code cell:
code 1
    print('Hello, world!')
 stream stdout
    Hello, world!
"""
```

And that's the proposed new format!

It can be converted to a notebook node (which will look the same as in the YAML example above):

[12]:
```
jupyter_format.deserialize(content)
```

[12]:
```
{'nbformat': 4,
 'nbformat_minor': 2,
 'metadata': {},
 'cells': [{'id': '12806a8b',
   'cell_type': 'markdown',
   'source': '# A Jupyter Notebook\n\nThis is a code cell:',
   'metadata': {}},
  {'id': '9c3b0b82',
   'cell_type': 'code',
   'metadata': {},
   'execution_count': 1,
   'source': "print('Hello, world!')",
   'outputs': [{'output_type': 'stream',
     'name': 'stdout',
     'text': 'Hello, world!'}]}]}
```

Just to make sure it is a valid Jupyter notebook node:

```
[13]: nbformat.validate(_)
```

2.5 Complementary Tools

Oftentimes cell outputs (e.g. plots) stored in notebooks make it hard to read and manipulate the text representation of such notebooks. They make it also hard to use with version control systems (e.g. Git).

The proposed new format has the same problem, outputs are still stored in the notebook file, right next to the code cells that generated them.

It is recommended to remove all outputs from a notebook before storing it in version control or before doing any manipulations with a text editor.

Outputs can be removed manually in the Jupyter user interface, but there are also tools to remove outputs programmatically:

- https://github.com/kynan/nbstripout
- https://github.com/choldgraf/nbclean
- https://github.com/toobaz/ipynb_output_filter

If you want to present your notebooks publicly, you often want to show the outputs to your audience, without them having to run the notebooks themselves. So do you have to store your outputs after all?

No! You can still store your notebooks without outputs and run your notebooks on a server that will re-create the outputs. One tool to do this is:

- https://nbsphinx.readthedocs.io/

This is a Sphinx[9] extension that can convert a bunch of Jupyter notebooks (and other source files) to HTML and PDF pages (and other output formats). This way you have the best of both worlds: No outputs in your (version controlled) notebook files, but full outputs in the public HTML (or PDF) version.

There are still some cases where you do want to store the outputs for some reason. Because of the outputs, it is hard to see the changes to the text/code content of the notebook with traditional tools like `diff`. But luckily, there is a tool that can make meaningful "diffs" for Jupyter notebooks:

- https://github.com/jupyter/nbdime

`doc/motivation.jupyter` ends here.

The following section was generated from `doc/spec.jupyter`

3 Specification

The exact specification is very much up for discussion!

For now, you can simply *convert* (page 9) an existing notebook and have a look at the resulting file in a text editor. And of course you can have a look at the source code.

If you have ideas for improving the (currently informal) specification or the example implementation, please open an issue or pull request at https://github.com/mgeier/jupyter-format/.

`doc/spec.jupyter` ends here.

[9] http://www.sphinx-doc.org/

4 Notebook format conversions with nbconvert

During the installation of jupyter_format (see *Installation* (page 2)), so-called "entry points" for nbconvert are configured automatically.

You can convert .ipynb notebooks to .jupyter notebooks with

```
python3 -m nbconvert --to jupyter my-old-notebook.ipynb
```

To convert a .jupyter notebook to any format supported by nbconvert, just append -from-jupyter to the desired format.

For example, you can convert a .jupyter notebook to the traditional .ipynb format:

```
python3 -m nbconvert --to ipynb-from-jupyter my-new-notebook.jupyter
```

Or you can convert a .jupyter file to an HTML file:

```
python3 -m nbconvert --to html-from-jupyter my-new-notebook.jupyter
```

Same for slides-from-jupyter, latex-from-jupyter, pdf-from-jupyter etc.

But enough for the theory, let's try it with this very notebook, shall we?

[1]:
```
!python3 -m nbconvert --to ipynb-from-jupyter nbconvert.jupyter --output=my-new-
↪notebook
```

```
[NbConvertApp] Converting notebook nbconvert.jupyter to ipynb-from-jupyter
[NbConvertApp] Writing 4326 bytes to my-new-notebook.ipynb
```

Just to make sure it is actually using Jupyter's JSON format, let's peek at the beginning of the file:

[2]:
```
!head my-new-notebook.ipynb
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "5e585e1d",
   "metadata": {
    "nbsphinx": "hidden"
   },
   "source": [
    "This notebook is part of the `jupyter_format` documentation:\n",
```

Here's a link to the new file for your perusal: my-new-notebook.ipynb.

Now let's convert this back to .jupyter:

[3]:
```
!python3 -m nbconvert --to jupyter my-new-notebook.ipynb
```

```
[NbConvertApp] Converting notebook my-new-notebook.ipynb to jupyter
[NbConvertApp] ERROR | Notebook JSON is invalid: Additional properties are not␣
↪allowed ('id' was unexpected)

Failed validating 'additionalProperties' in markdown_cell:

On instance['cells'][0]:
```

(continues on next page)

(continued from previous page)

```
{'cell_type': 'markdown',
 'id': '5e585e1d',
 'metadata': {'nbsphinx': 'hidden'},
 'source': 'This notebook is part of the `jupyter_format` documentation:\n'
           'htt...'}
[NbConvertApp] Writing 2686 bytes to my-new-notebook.jupyter
```

Again, we take a peek:

[4]:
```
!head my-new-notebook.jupyter
```

```
nbformat 4
nbformat_minor 2
markdown
    This notebook is part of the `jupyter_format` documentation:
    https://jupyter-format.readthedocs.io/.
 cell_metadata
    {
     "nbsphinx": "hidden"
    }
markdown
```

And a link for closer inspection: my-new-notebook.jupyter.

Finally, let's try to convert this `.jupyter` file to an HTML page:

[5]:
```
!python3 -m nbconvert --to html-from-jupyter my-new-notebook.jupyter
```

```
[NbConvertApp] Converting notebook my-new-notebook.jupyter to html-from-jupyter
[NbConvertApp] Writing 277891 bytes to my-new-notebook.html
```

[6]:
```
from IPython.display import IFrame
IFrame('my-new-notebook.html', width='100%', height=350)
```

[6]:
```
<IPython.lib.display.IFrame at 0x7fe58428a690>
```

And for completeness' sake, a link: my-new-notebook.html.

`doc/nbconvert.jupyter` ends here.

The following section was generated from `doc/api.jupyter`

This notebook contains mostly "raw" cells in reStructuredText[10] format, which are used to auto-generate the ...

5 API Documentation

`jupyter_format.`**`generate_lines`**`(`*`nb`*`)`

Generator yielding lines to be written to `.jupyter` files.

Each of the lines has a line separator at the end, therefore it can e.g. be used in `writelines()`[11].

Parameters

nb (*`nbformat.NotebookNode`*[12]) – A notebook node.

[10] http://docutils.sourceforge.net/rst.html
[11] https://docs.python.org/3/library/io.html#io.IOBase.writelines
[12] https://nbformat.readthedocs.io/en/latest/api.html#nbformat.NotebookNode

jupyter_format.**serialize**(*nb*)

Convert a Jupyter notebook to a string in `.jupyter` format.

Parameters
nb (*nbformat.NotebookNode*[13]) – A notebook node.

Returns
`.jupyter` file content.

Return type
str[14]

jupyter_format.**deserialize**(*source*)

Convert `.jupyter` string representation to Jupyter notebook.

Lines have to be terminated with `'\n'` (a.k.a. universal newlines[15] mode).

If *source* is an iterable, line terminators may be omitted.

Parameters
source (*str*[16] *or iterable of str*[17]) – Content of `.jupyter` file.

Returns
A notebook node.

Return type
nbformat.NotebookNode[18]

exception jupyter_format.**ParseError**

Exception that is thrown on errors during reading.

This reports the line number where the error occured.

5.1 Contents Manager

5.2 Exporters for `nbconvert`

class jupyter_format.exporters.**JupyterImportMixin**

Allow `*.jupyter` files as input to exporters.

This is used in all exporters as a "mixin" class.

from_file(*file*, *resources=None*, *jupyter_format=None*, ***kw*)

from_filename(*filename*, *resources=None*, ***kw*)

class jupyter_format.exporters.**JupyterExporter**(***kwargs: Any*[19])

Convert Jupyter notebooks to `.jupyter` format.

from_notebook_node(*nb*, *resources=None*, ***kw*)

The rest of the exporters are just the ones from *nbconvert.exporters*, enhanced with the *JupyterImportMixin*.

[13] https://nbformat.readthedocs.io/en/latest/api.html#nbformat.NotebookNode
[14] https://docs.python.org/3/library/stdtypes.html#str
[15] https://docs.python.org/3/glossary.html#term-universal-newlines
[16] https://docs.python.org/3/library/stdtypes.html#str
[17] https://docs.python.org/3/library/stdtypes.html#str
[18] https://nbformat.readthedocs.io/en/latest/api.html#nbformat.NotebookNode
[19] https://docs.python.org/3/library/typing.html#typing.Any

class jupyter_format.exporters.**ASCIIDocExporter**(**kwargs: *Any*[20])

class jupyter_format.exporters.**HTMLExporter**(**kwargs: *Any*[21])

See nbconvert.exporters.HTMLExporter[22].

class jupyter_format.exporters.**LatexExporter**(**kwargs: *Any*[23])

See nbconvert.exporters.LatexExporter[24].

class jupyter_format.exporters.**MarkdownExporter**(**kwargs: *Any*[25])

See nbconvert.exporters.MarkdownExporter[26].

class jupyter_format.exporters.**NotebookExporter**(**kwargs: *Any*[27])

See nbconvert.exporters.NotebookExporter[28].

class jupyter_format.exporters.**PDFExporter**(**kwargs: *Any*[29])

See nbconvert.exporters.PDFExporter[30].

class jupyter_format.exporters.**PythonExporter**(**kwargs: *Any*[31])

See nbconvert.exporters.PythonExporter[32].

class jupyter_format.exporters.**RSTExporter**(**kwargs: *Any*[33])

See nbconvert.exporters.RSTExporter[34].

class jupyter_format.exporters.**ScriptExporter**(**kwargs: *Any*[35])

class jupyter_format.exporters.**SlidesExporter**(**kwargs: *Any*[36])

See nbconvert.exporters.SlidesExporter[37].

class jupyter_format.exporters.**TemplateExporter**(**kwargs: *Any*[38])

See nbconvert.exporters.TemplateExporter[39].

5.3 Batch Conversion with `replace_all`

Script to recursively replace `.ipynb` with `.jupyter` files.

Usage:

```
python3 -m jupyter_format.replace_all --recursive --yes
```

WARNING: This deletes all original files!

Usage to apply this to the whole history of a Git branch:

[20] https://docs.python.org/3/library/typing.html#typing.Any
[21] https://docs.python.org/3/library/typing.html#typing.Any
[22] https://nbconvert.readthedocs.io/en/latest/api/exporters.html#nbconvert.exporters.HTMLExporter
[23] https://docs.python.org/3/library/typing.html#typing.Any
[24] https://nbconvert.readthedocs.io/en/latest/api/exporters.html#nbconvert.exporters.LatexExporter
[25] https://docs.python.org/3/library/typing.html#typing.Any
[26] https://nbconvert.readthedocs.io/en/latest/api/exporters.html#nbconvert.exporters.MarkdownExporter
[27] https://docs.python.org/3/library/typing.html#typing.Any
[28] https://nbconvert.readthedocs.io/en/latest/api/exporters.html#nbconvert.exporters.NotebookExporter
[29] https://docs.python.org/3/library/typing.html#typing.Any
[30] https://nbconvert.readthedocs.io/en/latest/api/exporters.html#nbconvert.exporters.PDFExporter
[31] https://docs.python.org/3/library/typing.html#typing.Any
[32] https://nbconvert.readthedocs.io/en/latest/api/exporters.html#nbconvert.exporters.PythonExporter
[33] https://docs.python.org/3/library/typing.html#typing.Any
[34] https://nbconvert.readthedocs.io/en/latest/api/exporters.html#nbconvert.exporters.RSTExporter
[35] https://docs.python.org/3/library/typing.html#typing.Any
[36] https://docs.python.org/3/library/typing.html#typing.Any
[37] https://nbconvert.readthedocs.io/en/latest/api/exporters.html#nbconvert.exporters.SlidesExporter
[38] https://docs.python.org/3/library/typing.html#typing.Any
[39] https://nbconvert.readthedocs.io/en/latest/api/exporters.html#nbconvert.exporters.TemplateExporter

```
git filter-branch --tree-filter "python3 -m jupyter_format.replace_all --
↪recursive --yes"
```

jupyter_format.replace_all.**ipynb_to_jupyter**(*path*)

Replace given `.ipynb` file with a `.jupyter` file.

WARNING: This deletes the original file!

Parameters

path (*os.PathLike*[40] *or str*[41]) – Path to `.ipynb` file.

jupyter_format.replace_all.**replace_all_recursive**(*start_dir*, *mapfunction=<class 'map'>*)

Replace all `.ipynb` files recursively.

WARNING: This deletes all original files!

Parameters

- **path** (*os.PathLike*[42] *or str*[43]) – Starting directory.
- **mapfunction** – `map()`[44]-like function that can be provided in order to enable parallelization.

doc/api.jupyter ends here.

doc/index.jupyter ends here.

[40] https://docs.python.org/3/library/os.html#os.PathLike
[41] https://docs.python.org/3/library/stdtypes.html#str
[42] https://docs.python.org/3/library/os.html#os.PathLike
[43] https://docs.python.org/3/library/stdtypes.html#str
[44] https://docs.python.org/3/library/functions.html#map

Python Module Index

j