<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citation</td>
<td>3</td>
</tr>
<tr>
<td>2 Badges</td>
<td></td>
</tr>
<tr>
<td>2.1 Getting Started</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Examples</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Notebook Conversion</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Metadata Tags</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Custom Export Configurations</td>
<td>18</td>
</tr>
<tr>
<td>2.6 Additional Tools</td>
<td>21</td>
</tr>
<tr>
<td>2.7 IPyPublish In The Wild</td>
<td>23</td>
</tr>
<tr>
<td>2.8 Acknowledgements</td>
<td>24</td>
</tr>
<tr>
<td>2.9 Package API</td>
<td>24</td>
</tr>
<tr>
<td>2.10 Releases</td>
<td>43</td>
</tr>
<tr>
<td>2.11 Example Export Configuration</td>
<td>46</td>
</tr>
<tr>
<td>2.12 Export Configuration Schema</td>
<td>48</td>
</tr>
<tr>
<td>2.13 Template Outline Example</td>
<td>51</td>
</tr>
<tr>
<td>2.14 Example of a Template Segment</td>
<td>54</td>
</tr>
<tr>
<td>3 Index and Search</td>
<td>57</td>
</tr>
</tbody>
</table>

Python Module Index  59
IP[y]:Publish

A package for creating and editing publication ready scientific reports and presentations, from Jupyter Notebooks.

**Attention:** A major API improvement occurred in v0.7.0. This has not changed the general user interface, but anyone using custom converter plugins will be required to update them (see Conversion of Plugins From Old API)

Combining features of the Jupyter Notebook, WYSIWYG editors and the Latex document preparation system, to provide a workflow for:

- Dynamic editing and visualisation of key document components (text, math, figures, tables, references, citations, etc).
- Combine document elements with dynamic (and reproducible) data exploration, analysis and visualisation.
- Supply meta formatting for document and code elements for precise control over the final document layout and typesetting.
- Output the same source document to different layouts and formats (pdf, html, presentation slides, etc).
Citation

Please cite if using IPyPublish.
BADGES

2.1 Getting Started

2.1.1 Installation

Using Conda is recommended for package management, in order to create self contained environments with specific versions of packages. The main external packages required are the Jupyter notebook and Pandoc (for conversion between file formats):

```
conda create --name ipyreport -c conda-forge jupyter pandoc
```

ipypublish can then be installed into this environment:

```
source activate ipyreport
pip install ipypublish
```

For converting to PDF, the TeX document preparation ecosystem is required, in particular\textit{latexmk}, which can be installed from:

- Linux: TeX Live
- macOS (OS X): MacTeX
- Windows: MikTeX

For helpful extensions to the notebooks core capabilities, see the Jupyter Notebook Extensions package:

```
conda install --name ipyreport jupyter_contrib_nbextensions
```

Additionally, a more extensive setup of useful packages (used to create the examples) are provided by the \texttt{anaconda distribution} which can be installed in to a new environment
2.1.2 Basic Conversion

The `nbpublish` script handles parsing the notebooks to nbconvert, with the appropriate converter.

```
conda create --name ipyreport anaconda
nbpublish -h
nbpublish -pdf -f latex_ipypublish_nocode path/to/notebook.ipynb
```

For a more detailed explanation see the `Notebook Conversion` section.

The `nbpresent` script handles serving reveal.js slides to a webbrowser.

```
nbpresent -h
nbpresent -f slides_ipypublish_nocode path/to/notebook.ipynb
```

Note that, for offline use, simply download the latest version of reveal.js [here](#), rename the entire folder to reveal.js and place it in the same folder as the converted .slides.html file. The slides can also be save to PDF my appending `pdf-export` to the url (see [here](#) for details).

2.1.3 Troubleshooting

For installation issues, Travis CI is used to automatically test updates against python 2.7 and 3.6, for both Linux and OSX. Therefore, to troubleshoot any installation/run issues, it is best to first look at the [travis config and travis test runs](#) for working configurations.

The `requirements-lock.txt` file can also be used to provide exact versions of working package dependencies.

For conversion issues, for both `nbpublish` and `nbpresent`, detailed log messages of the run are output to both the console and file (default path: converted/notebook_name.nbpub.log). To debug conversions, use the `--log-level debug` and `--pdf-debug` flags. If there is still an error, please raise an issue on the [GitHub repository](#), including the run environment and the log file.

2.2 Examples

For an example of the potential input/output:

- Example.ipynb
- Example.pdf
- Example.html
- Example.slides.html

Or, for a practical example of the ipypublish capability, see these documents on Atomic 3D Visualisation:

- Notebook,
- PDF,
- HTML
- Reveal.JS slideshow.
2.3 Notebook Conversion

2.3.1 Setting up a Notebook

For improved latex/pdf output, `ipynb_latex_setup.py` contains import and setup code for the notebook and a number of common packages and functions, including:

- numpy, matplotlib, pandas, sympy, ...

- `images_hconcat`, `images_vconcat` and `images_gridconcat` functions, which use the PIL/Pillow package to create a single image from multiple images (with specified arrangement)

To use this script, in the first cell of a notebook, insert:

```python
from ipypublish.scripts.ipynb_latex_setup import *
```

It is recommended that you also set this cell as an initialisation cell (i.e. have "init_cell": true in the metadata)

For existing notebooks: the `nb_ipypublish_all` and `nb_ipypublish_nocode` converters (see below) can be helpful for outputting a notebook, with identical content to that input, but with default metatags defining how content is to be output.

2.3.2 Converting Notebooks

The `nb_publish` script handles parsing the notebooks to nbconvert, with the appropriate converter. To see all options for this script:

```
nb_publish -h
```

For example, to convert the Example.ipynb notebook directly to pdf:

```
nb_publish -pdf example/notebooks/Example.ipynb
```

If a folder is input, then the .ipynb files it contains are processed and combined in ‘natural’ sorted order, i.e. 2_name.ipynb before 10_name.ipynb. By default, notebooks beginning '_' are ignored.

All available converters are also listed by `nb_publish --list-exporters`. Three of note are:

- `latex_ipypublish_main` is the default and converts cells to latex according to metadata tags on an ‘opt in’ basis. Note that, for this converter, no code cells or output will appear in the final tex/pdf document unless they have a suitable `ipub metadata tag`.

- `html_ipypublish_main` converts the entire notebook(s) to html and adds a table of contents sidebar and a button to toggle input code and output cells visible/hidden, with latex citations and references resolved.

- `slides_ipypublish_main` converts the notebook to reveal.js slides, with latex citations and references resolved and slide partitioning by markdown headers. See the Live Slideshows section for using nbpresent to serve these slides to a webbrowser.

- The all and nocode variants of these converters preprocess a copy of the notebook, to add default metadata tags to the notebook and all cells, such that all output is rendered (with or without the code)

The current nbconvert --to pdf does not correctly resolve references and citations (since it copies the files to a temporary directory). Therefore nbconvert is only used for the initial nbconvert --to latex phase, followed by using latexmk to create the pdf and correctly resolve everything. To convert your own notebook to PDF for the first time, a good route would be to use:
2.3.3 The IPyPublish Defaults

The ipypublish ‘main’ converters are designed with the goal of creating a single notebook, which may contain lots of exploratory code/outputs, mixed with final output, and that can be output as both a document (latex/pdf or html) and a presentation (reveal.js). The logic behind the default output is then:

- For documents: all headings and body text is generally required, but only a certain subset of code/output
- For slides: all headings are required, but most of the body text will be left out and substitutied with ‘abbreviated’ versions, and only a certain subset of code/output.

This leads to the following logic flow (discussed further in the Metadata Tags section):

latex_ipypublish_main and html_ipypublish_main

- all cells: bypass “ignore” and “slideonly” tags
- markdown cells: include all
- code cells (input): only include if the “code” tag is present
- code cells (output): only include if the following tags are present
  - “figure” for png/svg/pdf/jpeg or html (html only)
  - “table” or “equation” for latex or html (html only)
  - “mkdown” for markdown text
  - “text” for plain text

slides_ipypublish_main

- all cells: bypass “ignore”
- markdown cells: are first split into header (beggining #)/non-header components
  - headers: include all
  - non-headers: only include if “slide” tag
- code cells (input): only include if the “code” tag is present
- code cells (output): only include if the following tags are present
  - “figure” for png/svg/pdf/jpeg/html
  - “table” or “equation” for latex/html
  - “mkdown” for markdown text
  - “text” for plain text
Packages, such as pandas and matplotlib, use jupyter notebooks rich representation mechanics to store a single output in multiple formats. nbconvert (and hence ipypublish) then selects only the highest priority (compatible) format to be output. This allows, for example, for pandas DataFrames to be output as latex tables in latex documents and html tables in html documents/slides.

### 2.3.4 Simple Customisation of Outputs

To customise the output of the above defaults, simply download one of:

- latex_ipypublish_all.json.
- html_ipypublish_all.json.
- slides_ipypublish_all.json.

Then alter the cell_defaults and nb_defaults sections, and run:

```bash
nbpublish -f path/to/new_config.json input.ipynb
```

### 2.4 Metadata Tags

#### 2.4.1 Introduction

All additional information, used to specify how a particular notebook/cell/output will be represented, when converted, is stored in the metadata under:

```json
{
   "ipub": {}
}
```

There are three levels of metadata:

- For notebook level: in the Jupyter Notebook Toolbar go to Edit -> Edit Notebook Metadata
- For cell level: in the Jupyter Notebook Toolbar go to View -> Cell Toolbar -> Edit Metadata and a button will appear above each cell.
- For output level: using `IPython.display.display(obj,metadata={"ipub":{}})`, you can set metadata specific to a certain output. Options set at the output level will override options set at the cell level. For an example of this, download and run the MultiOutput_Example.ipynb.

**Important:** setting a value to "value":{} is the same as "value":false so, if you are not setting additional options, use "value":true.

**See also:**
The Notebook file format

#### 2.4.2 Visualising Metadata

New in version 0.7.0: To view all the metadata in a notebook, you can now use the python_with_meta exporter.
nbpublish -f python_with_meta example.ipynb

This will produce a standard python file, with metadata commented by `#~~` and each cell beginning with `#%%` (known as the percent format):

```
#~~ language_info:
   #~~ name: python
   #~~ nbconvert_exporter: python
   #~~ pygments_lexer: ipython3
   #~~ version: 3.6.1

#%% [markdown]
#~~ {}
# % Document Title
#%
#~~ ipub:
#~~ figure:
#~~ title: A nice picture.
#~~ label: fig:example
#~~ placement: '!bh'
Image('example.jpg',height=400)
```

Alternatively, you can use the excellent `jupytext` package, to convert between a notebook and percent format. Simply add, this section to the notebook-level metadata:

```
{
   "jupytext": {
      "metadata_filter": {
         "notebook": "ipub"
      }
   }
}
```

and run:

```
jupytext --to py:percent notebook.py
```

Then, after altering the python file, run:

```
jupytext --to notebook notebook.py  # overwrite notebook.ipynb (remove_outputs)
jupytext --to notebook --update notebook.py  # update notebook.ipynb (preserve_outputs)
```

The percent format can be utilised in IDEs, such as Spyder, Atom, PyCharm, and VS Code, to run individual cells:

### 2.4.3 Document Level

**Language**

To change the **language** of the document:

```
{
   "ipub": {
      "jupytext": {
         "metadata_filter": {
            "notebook": "ipub"
         }
      }
   }
}
```
Fig. 1: Running Python File in VS Code

```json
  {  
"language": "french"  
  }
}
```

where the language can be any specified in the babel package.

**Bibliography**

To specify where the bibliography is and choose a style:

```json
  {  
"ipub": {  
"bibliography": "path/to/bibliograph.bib",  
"bibstyle": "unsrtnat",  
"biboptions": ["super", "sort"],  
}  
}
```

- The path can be absolute or relative.
- The **bibstyle** must be a natbib stylename

New in version 0.7.1: The **biboptions** is a list of options to parse to natbib. The default is: [“numbers”, “square”, “super”, “sort&compress”], and some common options are:

- **round**: (default) for round parentheses;
- **square**: for square brackets;
- **curly**: for curly braces;

---

2.4. Metadata Tags 11
– *angle*: for angle brackets;
– *colon*: (default) to separate multiple citations with colons;
– *comma*: to use commas as separators;
– *authoryear*: (default) for author-year citations;
– *numbers*: for numerical citations;
– *super*: for superscripted numerical citations, as in Nature;
– *sort*: orders multiple citations into the sequence in which they appear in the list of references;
– *sort&compress*: as sort but in addition multiple numerical citations are compressed if possible (e.g. 3-6, 15);
– *longnamesfirst*: makes the first citation of any reference the equivalent of the starred variant (full author list) and subsequent citations normal (abbreviated list);

**Title Page**

For *titlepage*, enter in notebook metadata:

```json
{
  "ipub": {
    "titlepage": {
      "author": "Authors Name",
      "email": "authors@email.com",
      "supervisors": [
        "First Supervisor",
        "Second Supervisor"
      ],
      "title": "Main-Title",
      "subtitle": "Sub-Title",
      "tagline": "A tagline for the report."
    },
    "institution": [
      "Institution1",
      "Institution2"
    ],
    "logo": "path/to/logo_example.png"
  }
}
```

• all keys are optional
• if there is no title, then the notebook filename will be used
• if nbpublish.py is called on a folder, then the meta data from the first notebook will be used
• logo should be the path (absolute or relative) to a logo image file

**Contents Tables**

To control the output of *contents tables*: 
To override the default placement of figures and tables:

```
{
  "ipub": {
    "figure": {
      "placement": "!bp"
    },
    "table": {
      "placement": "!bp"
    }
  }
}
```

See Positioning_images_and_tables for placement options.

### 2.4.4 Cell/Output Level

#### Ignore

To ignore any cell for all outputs:

```
{
  "ipub": {
    "ignore" : true
  }
}
```

To mark any cell as for output to slides only:

```
{
  "ipub": {
    "slideonly" : true
  }
}
```

#### Code Block

To output a code block:

```
{
  "ipub": {
    "code": {
      "format": {},
      "asfloat": true,
      "caption": "",
    }
  }
}
```
all extra tags are optional:

- `format` can contain any keywords related to the latex Listings package (such as syntax highlighting colors).
- `asfloat` constitutes whether the code is wrapped in a codecell (float) environment or is inline.
- all other tags work the same as figure (below).

### Output Text

To output text produced by the code (e.g. via the `print` command):

```json
{
  "ipub": {
    "text": {
      "format": {
        "basicstyle": "\small"
      },
      "asfloat": true,
      "caption": "",
      "label": "code:example_sym",
      "widefigure": false,
      "placement": "H",
      "use_ansi": false
    }
  }
}
```

all extra tags are optional:

- `format` can contain any keywords related to the latex Listings package (such as syntax highlighting colors).
  N.B. in place of `\use`. 
- `asfloat` constitutes whether the code is wrapped in a codecell (float) environment or is inline.
- if `use_ansi` is true then, instead of stripping ansi colors in latex output, they will be converted to latex, wrapped in `%` characters and the listings option `escapechar=%` set.
- all other tags work the same as figure (below).

### Output Figures

For figures (i.e. any graphics output by the code), enter in cell metadata:

```json
{
  "ipub": {
    "figure": {
      "caption": "Figure caption.",
      "label": "fig:flabel",
      "placement": "H",
    }
  }
}
```

(continues on next page)
• all tags are optional
• height/width correspond to the fraction of the page height/width, only one should be used (aspect ratio will be maintained automatically)
• placement is optional and constitutes using a placement arguments for the figure (see Positioning_images_and_tables).

```
\begin{figure}[H]
```

• widefigure is optional and constitutes expanding the figure to the page width (placement arguments will then be ignored)

```
\begin{figure*}
```

## Output Tables

For tables (e.g. those output by pandas), enter in cell metadata:

```
{
  "ipub": {
    "table": {
      "caption": "Table caption.",
      "label": "tbl:tlabel",
      "placement": "H",
      "alternate": "gray!20"
    }
  }
}
```

• caption and label are optional
• placement is optional and constitutes using a placement arguments for the table (see Positioning_images_and_tables).

```
\begin{table}[H]
```

• alternate is optional and constitutes using alternating colors for the table rows (see https://tex.stackexchange.com/a/5365/107738).

```
\rowcolors{2}{gray!25}{white}
```

• if tables exceed the text width, in latex, they will be shrunk to fit

## Output Equations

For equations (e.g. those output by sympy), enter in cell metadata:

```
2.4. Metadata Tags 15
```
• environment is optional and can be ‘none’ or any of those available in \texttt{amsmath}; ‘equation’, ‘align’, ‘multiline’, ‘gather’, or their * variants. Additionally, ‘breqn’ or ‘breqn*’ will select the experimental \texttt{breqn} environment to \textit{smart} wrap long equations.

• label is optional and will only be used if the equation is in an environment

\section*{Controlling Slides}

For \texttt{slide output}:

\begin{verbatim}
{  "ipub": {  "slide": true  }}
\end{verbatim}

• the value of slide can be true, “new” (to indicate the start of a new slide) or “notes”

\section*{Object Output Formats}

The format of the Jupyter Notebook (.ipynb) file allows for the storage of a single output in multiple formats. This is taken advantage of by packages such as matplotlib and pandas, etc to store a figure/table in both latex and html formats, which can then be selected by ipypublish based on the document type required.

Sometimes a user may wish to have greater control over the output format and/or which output types are to be stored. It is possible to achieve this via the Jupyter \texttt{display} function. For example, if we wanted to display a pandas.DataFrame table without the index column, such that it can be output to both a pdf and html document:

\begin{verbatim}
from IPython.display import display
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.random((3, 3)))
latex = df.to_latex(index=False)
html = df.to_html(index=False)
display({'text/latex': latex,  'text/html': html}, raw=True)
\end{verbatim}

If you wish to create your own object with multiple output formats, you should create a class with multiple \_\_repr\_*\_() methods (as described here):

\begin{verbatim}
class MyObject(object):
    def \_\_init\_\_(self, text):
        self.text = text

    def \_\_repr\_latex\_\_(self):
        return "textbf\{" + self.text + "}\}
\end{verbatim}

(continues on next page)
Captions in a Markdown cell

Especially for long captions, it would be preferred that they can be viewed and edited in a notebook Markdown cell, rather than hidden in the metadata. This can be achieved using the default ipypublish converters:

If a **markdown cell** or **code cell with latex/text output** has the metadata tag:

```json
{
    "ipub": {
        "caption": "fig:example_mpl"
    }
}
```

Then, during the the postprocessor stage, this cell will be removed from the notebook object, and its text stored as a resource;

• the cell’s text is the first paragraph of the markdown string, i.e. nothing after a newline (\n)
• if there are multiple instance of the same caption name, then only the last instance will be stored

During the jinja templating, if a **figure, table or code** cell has a label matching any stored caption name, for example:

```json
{
    "ipub": {
        "figure": {
            "caption": "",
            "label": "fig:example_mpl"
        }
    }
}
```

Then its caption will be overridden with the stored text.

**Embedding Interactive HTML**

Packages built on **IPywidgets**, like PythreeJS, Pandas3JS and the excellent **IPyvolume**, are making it increasingly easier to render complex, interactive html in the notebook. IPywidgets offers a **save notebook with widgets** feature, however, this can greatly increase the size of the notebook.

A better solution, recently offered, is to save a **html snippet** of the current widget state to file and embed it into the html/slides output as an iframe. This is also particularly useful in reveal.js slides, since the iframe content can be lazy loaded. To embed html, use the **embed_html** tag:

```json
{
    "ipub": {
        "embed_html": {
            "filepath": "path/to/file.html",
            "other_files": ["path/to/file.js"],
            "url": "https://path/to/url.html",
            "width":0.5,
            "height":0.5
        }
    }
}
```
If the cell already contains an output, then this tag will create/overwrite the first output’s “text/html” type. This allows for a single notebook cell with a static image of the widget in the output, and a path to the embed html in the metadata so that a) if you export to latex/pdf, you get the static image or b) if you export to html/reveal slides, you get the html.

- use either filepath or url
- other_files are files required by the html file (e.g. javascript libraries). These files will be copied to the the same folder as the html
- width/height refers to the fraction of the viewspace used (e.g. 0.5 width -> 50vw and 0.5 height -> 50vh)

An example of how this works is in the Example.ipynb, and the Example.html and Example.slides.html outputs.

### 2.5 Custom Export Configurations

The simplest application of this, would be to copy `latex_ipypublish_all.json` (or the html/slides variants) and make changes to the `cell_defaults` and `nb_defaults` dictionaries, to suit your output needs, then run:

```
nbconvert -f path/to/latex_ipypublish_all.json notebook.ipynb
```

#### 2.5.1 The Conversion Process

iPyPublish uses export configuration files to control how the Notebook(s) will be exported. As shown in the figure below, they define two key components:

1. The export class, and its associated pre-processors and filter functions.
2. The jinja template outline and segments to be inserted into it.

![Fig. 2: iPyPublish Conversion Process](image)

This process extends `nbconvert` in a number of ways:

- Merging of notebooks is handled automatically
- Numerous additional `ipypublish.preprocessors` and `ipypublish.filters` are supplied.
- `jinja` templates are constructed via segment insertions, into a skeleton (outline) template, rather than by inheritance only. This allows for greater control and modularity in their construction.
- The use of `latexmk` with XeLaTeX to convert TeX to PDF, and correct resolution of file references and citations.

#### 2.5.2 The Configuration File Format

The configuration file is a JSON file, with a validation schema given in `Export Configuration Schema`. Below is a minimal example:
Exporter Class

In line 6, we define the exporter class, which can be any class in the python environment namespace that inherits from `nbconvert.exporters.Exporter`.

Exporters can be parsed any number of preprocessors (inheriting from `nbconvert.preprocessors.Preprocessor`), which act on the notebook in the order supplied.

The `args` field is used to set any configurable `traitlets` the class exposes. Two special placeholders are available:

- `${meta_path}` will be set dynamically as the path to the (primary) ipynb file, containing the document level meta-data.
- `${files_path}` will be set dynamically as the path to the folder where, additional files (such as internal images) will be output to.

Filters provide functions or classes to transform particular content of the notebook, and are parsed to the `jinja` templating engine.
See also:

- The classes available natively in nbconvert: `nbconvert.exporters`, `nbconvert.preprocessors`, `nbconvert.filters`.
- How Filters are used in jinja.

Template Construction

In line 22, we define how to construct the jinja template. The `outline` key defines the path to an outline template, such as in Template Outline Example.

Changed in version 0.8.0: The outline file is now a jinja template, instead of a JSON file

This template file can be a full jinja template file, extending an existing nbconvert template, but may optionally contain 'placeholders' (of the form `@ipubreplace{below}{key_name}`) that can be replaced by injecting zero or more segments into them. The first option states whether segment injections are appended above or below previous injections, and the second option defines the key for that segment.

This approach allows independent aspects of the document to be stored separately then pieced together in the desired manner. For example, the segment file in Example of a Template Segment defines only parts of the document which control how the bibliography is constructed. This could be removed or replaced by a custom export configuration. Similarly, input and output prompts can be added/removed in html documents.

Segments are applied in the order they are defined, and appended above or below existing content, as defined by the placeholder. For example, these segments:

```json
[
    {
        "notebook_input_markdown_pre": "<div class='inner'>",
        "notebook_input_markdown": " test",
        "notebook_input_markdown_post": "</div>",
    },
    {
        "notebook_input_markdown_pre": "<div class='outer'>",
        "notebook_input_markdown_post": "</div>",
    }
]
```

applied to this template outline:

```jinja
{% block markdowncell scoped %}
@ipubreplace{above}{notebook_input_markdown_pre}
@ipubreplace{below}{notebook_input_markdown}
@ipubreplace{below}{notebook_input_markdown_post}
{% endblock markdowncell %}
```

will result in a template containing:

```jinja
{% block markdowncell scoped %}
<div class='outer'>
    <div class='inner'>
        test
    </div>
</div>
{% endblock markdowncell %}
```

Segment configuration files also have an optional `overwrite` key, which define segments that overwrite any previously defined content in that section.
See also:

- The jinja documentation on Template Designer Documentation
- The nbconvert documentation on Customizing nbconvert

### 2.5.3 Loading Custom Configurations

Custom configurations can be parsed directly to nbpublish:

```
nbpublish -f path/to/configs/export_config.json notebook.ipynb
```

Or used as a key, by providing nbpublish with additional folders to scan (in addition to the `ipypublish.export_plugins` module folder):

```
nbpublish -ep path/to/configs -f export_config notebook.ipynb
```

### 2.5.4 Conversion of Plugins From Old API

The old style export plugins (defined as python scripts) can be converted to the new JSON style, using the `ipypublish.port_api.plugin_to_json.convert_to_json()` function.

The old style template segment dictionaries (defined as python scripts) can be converted to the new JSON style, using the `ipypublish.port_api.tpl_dct_to_json.py_to_json()` function.

### 2.6 Additional Tools

#### 2.6.1 Citations and Bibliography

Using Zotero’s Firefox plugin and Zotero Better Bibtex for:

- automated .bib file updating
- drag and drop cite keys `cite{kirkeminde_thermodynamic_2012}`
- `latexmk -bibtex -pdf` (in nbpublish.py) handles creation of the bibliography
- `\usepackage{doi}` turns the DOI numbers into url links
  - in Zotero-Better-Bibtex you have the option set to only export DOI, if both DOI and URL are present.

Can use:

```
<cite data-cite="kirkeminde_thermodynamic_2012">(Kirkeminde, 2012)</cite>
```

to make it look better in html, but not specifically available for drag and drop in Zotero.

#### 2.6.2 Live Slideshows

The Reveal.js - Jupyter/IPython Slideshow Extension (RISE) notebook extension offers rendering as a Reveal.js-based slideshow, where you can execute code or show to the audience whatever you can show/do inside the notebook itself! Click on the image to see a demo:
2.6.3 Working With External Data

A goal for scientific publishing is automated reproducibility of analyses, which the Jupyter notebook excels at. But, more than that, it should be possible to efficiently reproduce the analysis with different data sets. This entails having **one point of access** to a data set within the notebook, rather than having copy-pasted data into variables, i.e. this:

```python
data = read_in_data('data_key')
variable1 = data.key1
variable2 = data.key2
...```

rather than this:

```python
variable1 = 12345
variable2 = 'something'
...```

The best-practice for accessing heirarchical data (in my opinion) is to use the JSON format (as long as the data isn’t relational), because it is:

- applicable for any data structure
- lightweight and easy to read and edit
- has a simple read/write mapping to python objects (using json)
• widely used (especially in web technologies)

A good way to store multiple bits of JSON data is in a MongoDB and accessing it via pymongo. This will also make it easy to move all the data to a cloud server at a later time, if required.

```
conda install pymongo
```

But, if the data is coming from files output from different simulation or experimental code, where the user has no control of the output format. Then writing JSON parsers may be the way to go, and this is where jsonextended comes in, which implements:

• a lightweight plugin system to define bespoke classes for parsing different file extensions and data types.
• a ‘lazy loader’ for treating an entire directory structure as a nested dictionary.

For example:

```
from jsonextended import plugins, edict
plugins.load_plugins_dir('path/to/folder_of_parsers','parsers')
data = edict.LazyLoad('path/to/data')
variable1 = data.folder1.file1_json.key1
variable2 = data[‘folder1’,’file1.json’,’key2’]
variable3 = data[‘folder1’,’file2.csv’,’key1’]
variable4 = data[‘folder2’,’subfolder1’,’file3.other’,’key1’]
...
```

If you are dealing with numerical data arrays which are too large to be loaded directly in to memory, then the h5py interface to the HDF5 binary data format, allows for the manipulation of even multi-terabyte datasets stored on disk, as if they were real NumPy arrays. These files are also supported by jsonextended lazy loading.

### 2.6.4 Miscellaneous

I also use the Firefox Split Panel extension to view the \{name\} _viewpdf.html page and monitor changes to the pdf. bookbook is another package with some conversion capabilities.

### 2.7 IPyPublish In The Wild

• University of Oslo:
  – 2018 Nordic eScience Globalisation Initiative (NeGI): Climate science at high latitudes: Modelling and model evaluation
  – 2019 Research Bazarr: Publication ready scientific reports and presentations with Jupyter notebooks

• Universidad del Dosario
  – 2019 Bogota Experimental Economics Workshop: Intro to Dynamic documents
  – Lead by Berkeley Initiative for Transparency in the Social Sciences
  – See also Experimetrics-BITSS-Workshop

• Generating Software Tests eBook
2.8 Acknowledgements

IPyPublish is built as an extension to nbconvert.
I also took strong influence from:

- Julius Schulz
- Dan Mackinlay
- Notebook concatenation was adapted from nbconvert issue#253

2.9 Package API

2.9.1 ipypublish package

Subpackages

ipypublish.convert package

Submodules

ipypublish.convert.config_manager module

- ipypublish.convert.config_manager.create_exporter_cls(class_str)
  dynamically load export class
- ipypublish.convert.config_manager.get_export_config_path(export_key, config_folder_paths=())
  we search for a plugin name, which matches the supplied plugin name
- ipypublish.convert.config_manager.get_export_extension(export_config_path)
  return the file extension of the exporter class
- ipypublish.convert.config_manager.iter_all_export_infos(config_folder_paths=(), regex='*.json')
  iterate through all export configuration and yield a dict of info
- ipypublish.convert.config_manager.iter_all_export_paths(config_folder_paths=(), regex='*.json')
  we iterate through all json files in the supplied plugin_folder_paths, and then in the export_plugins folder
- ipypublish.convert.config_manager.load_export_config(export_config_path)
  load the export configuration
- ipypublish.convert.config_manager.load_template(template_dict)
- ipypublish.convert.config_manager.str_to_jinja(template_str)

ipypublish.convert.main module

- ipypublish.convert.main.create_config(exporter_data, replacements)
- ipypublish.convert.main.dict_to_config(config, unflatten=True)
- ipypublish.convert.main.export_notebook(final_nb, exporter_cls, config, jinja_template)
convert one or more Jupyter notebooks to a published format
paths can be string of an existing file or folder, or a pathlib.Path like object
all files linked in the documents are placed into a single folder

Parameters

- **ipynb_path** – notebook file or directory
- **outformat** (`str`) – output format to use
- **outpath** (`str` or `pathlib.Path`) – path to output converted files
- **dump_files** (`bool`) – whether to write files from nbconvert (images, etc) to outpath
- **ignore_prefix** (`str`) – ignore ipynb files with this prefix
- **clear_existing** (`str`) – whether to clear existing external files in outpath folder
- **create_pdf** (`bool`) – whether to convert to pdf (if converting to latex)
- **pdf_in_temp** (`bool`) – whether to run pdf conversion in a temporary folder
- **pdf_debug** (`bool`) – if True, run latexmk in interactive mode
- **dry_run** (`bool`) – if True, do not create any files

Returns

- **outpath** (`str`) – path to output file
- **exporter** (`nbconvert.exporters.Exporter`) – the exporter used

Module contents

*ipypublish.export_plugins package*

Module contents

*ipypublish.filters package*

Submodules

*ipypublish.filters.ansi_listings module*

Filters for processing ANSI colors within Jinja templates.

Convert ANSI colors to LaTeX colors.
Parameters

- **text** (*str*) – Text containing ANSI colors to convert to LaTeX
- **escapechar** (*str*) – escape character

Examples

```python
>>> print(ansi2listings('[32mFolder[0m("subdir1")'))
%\textcolor{ansi-green}{Folder}%("subdir1")
```

```python
>>> print(ansi2listings('[1;32mFolder[0m("subdir1")'))
%\textcolor{ansi-green-intense}{\textbf{Folder}}%("subdir1")
```

```python
>>> print(ansi2listings('[38;2;10;10;10mFolder[0m("subdir1")'))
%\def\tcRGB{\textcolor[RGB]}\expandafter{tcRGB}\expandafter{\detokenize{10,10,10}}
→{\textcolor{ansi-green}{Folder}%("subdir1")}
```

**ipypublish.filters.filters module**

**ipypublish.filters.filters.create_key** (*input, **kwargs*)
create sanitized key string which only contains lowercase letters, (semi)colons as c, underscores as u and numbers as roman numerals in this way the keys with different input should mainly be unique

```python
>>> create_key('fig:A_10name56')
'figcauxnamelvi'
```

**ipypublish.filters.filters.dict_to_kwds** (*inobject, kwstr=", overwrite=True*)
convert a dictionary to a string of keywords, or, if a list, a string of options append to an existing options string (without duplication)

Parameters

- **dict** (*dict*) –
- **kwstr** (*str*) – initial keyword string
- **overwrite** (*bool*) – overwrite the option, if it already exists with a different value

Examples

```python
>>> dict_to_kwds({'a':1,'c':3},'a=1,b=2')
'a=1,b=2,c=3'
```

```python
>>> dict_to_kwds(['a', 'c'], 'a,b')
'a,b,c'
```

**ipypublish.filters.filters.first_para** (*input, **kwargs*)
going only text before a n (i.e. the fist paragraph)

**ipypublish.filters.filters.is_equation** (*text*)

**ipypublish.filters.filters.remove_dollars** (*input, **kwargs*)
remove dollars from start/end of file

**ipypublish.filters.filters.strip_ext** (*path*)
```python
ipypublish.filters.filters.wrap_latex(input, max_length=75, **kwargs)
```

```
ipypublish.filters.meta_to_yaml module

ipypublish.filters.meta_to_yaml.meta_to_yaml(metadata, comment='#~~ ')
convert metadata json to yaml

Parameters

- **metadata (nbformat.notebooknode.NotebookNode)** –
- **comment** (str) – append to the start of each line

Returns str
Return type str

```

```
ipypublish.filters.meta_to_yaml.recurse_convert(node)
convert notebook node to dict
```

```
ipypublish.filters.replace_string module

ipypublish.filters.replace_string.replace_string(line, find, replace)
```

Module contents

```
ipypublish.frontend package

Submodules

ipypublish.frontend.nbpresent module

ipypublish.frontend.nbpresent.nbpresent(inpath, outformat='slides_standard', outpath=None, dump_files=True, ignore_prefix='_', clear_files=False, log_level='INFO', dry_run=False, export_paths=())
load reveal.js slides as a web server, converting from ipynb first if path extension is .ipynb

Parameters

- **inpath (str)** – path to html or ipynb file
- **outformat (str)** – conversion format to use
- **outpath (str or pathlib.Path)** – path to output converted files
- **dump_files (bool)** – whether to write files from nbconvert (images, etc) to outpath
- **clear_files (str)** – whether to clear existing external files in outpath folder
- **ignore_prefix (str)** – ignore ipynb files with this prefix
- **log_level (str)** – the logging level (debug, info, critical, …)

```
ipypublish.frontend.nbpresent.run(sys_args=None)
```
ipypublish.frontend.nbpublish module

```python
ipypublish.frontend.nbpublish(ipynb_path, outformat='latex_ipypublish_main', outpath=None, dump_files=True, ignore_prefix='_', clear_files=False, create_pdf=False, pdf_in_temp=False, pdf_debug=False, log_level='INFO', dry_run=False, export_paths=())
```

convert one or more Jupyter notebooks to a published format

paths can be string of an existing file or folder, or a pathlib.Path like object

**Parameters**

- **ipynb_path** – notebook file or directory
- **outformat** (*str*) – output format to use
- **outpath** (*str or pathlib.Path*) – path to output converted files
- **dump_files** (*bool*) – write files from nbconvert (containing images, etc) to outpath
- **ignore_prefix** (*str*) – ignore ipynb files with this prefix
- **clear_files** (*str*) – whether to clear existing external files in outpath folder
- **create_pdf** (*bool*) – convert to pdf (if converting to latex)
- **pdf_in_temp** (*bool*) – run pdf conversion in a temporary folder and only copy back the pdf file
- **pdf_debug** (*bool*) – run latexmk in interactive mode
- **log_level** (*str*) – the logging level (debug, info, critical, ...)

ipypublish.frontend.nbpublish.run(sys_args=None)

ipypublish.frontend.shared module

```python
class ipypublish.frontend.shared.CustomFormatter(prog, indent_increment=2, max_help_position=24, width=None)
Bases: argparse.ArgumentDefaultsHelpFormatter, argparse.RawDescriptionHelpFormatter

class ipypublish.frontend.shared.CustomParser(prog=None, usage=None, description=None, epilog=None, parents=[], formatter_class=<class 'argparse.HelpFormatter'>, prefix_chars='-', fromfile_prefix_chars=None, argument_default=None, conflict_handler='error', add_help=True, allow_abbrev=True)
Bases: argparse.ArgumentParser

class ipypublish.frontend.shared.CustomParser

class ipypublish.frontend.shared.get_parser(**kwargs)
```
ipypublish.frontend.shared.get_plugin_str(plugin_folderPaths)
return string listing all available export configurations

ipypublish.frontend.shared.parse_options(sys_args, program)

Module contents

ipypublish.port_api package

Submodules

ipypublish.port_api.plugin_to_json module

a module to convert between the old (Python script) plugin format, and the new (JSON) one

ipypublish.port_api.plugin_to_json.assess_syntax(path)

ipypublish.port_api.plugin_to_json.ast_to_json(item, imported, assignments)
recursively convert ast items to json friendly values

ipypublish.port_api.plugin_to_json.convert_config(config, exporter_class, allow_other)
convert config into required exporter format

ipypublish.port_api.plugin_to_json.convert_dict(dct, imported, assignments)
recurse through and replace keys

ipypublish.port_api.plugin_to_json.convert_oformat(oformat)

ipypublish.port_api.plugin_to_json.convert_to_json(path, outpath=None, ignore_other=False)
Set docstring here.

Parameters

• path (str) – input module path
• outpath=None (str or None) – if set, output json to this path
• ignore_other (bool) – whether to ignore arguments in config, which do not relate to preprocessors

ipypublish.port_api.plugin_to_json.create_json(docstring, imported, assignments, allow_other=True)
Set docstring here.

Parameters

• docstring (str) – the doc string of the module
• imported (dict) – imported classes
• assignments (dict) – assigned values (i.e. ‘a = b’)
• allow_other (bool) – whether to allow arguments in config, which do not relate to preprocessors

ipypublish.port_api.plugin_to_json.replace_template_path(path)
replace original template path with new dict

2.9. Package API
**ipypublish Documentation, Release 0.8.0**

**ipypublish.port_api.tpl_dct_to_json module**

A module to convert between the old (Python script) segment format, and the new (JSON) one.

*ipypublish.port_api.tpl_dct_to_json.*

**assess_syntax**(path)

*ipypublish.port_api.tpl_dct_to_json.*

**py_to_json**(path, outpath=None)

**Module contents**

**ipypublish.preprocessors package**

**Submodules**

**ipypublish.preprocessors.crop_cells module**

```python
class ipypublish.preprocessors.crop_cells.CropCells(**kw):
    Bases: nbconvert.preprocessors.base.Preprocessor
    A preprocessor to crop the notebook cells from <start> to <end>

    end
      An int trait.

    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.

    start
      An int trait.
```

**ipypublish.preprocessors.latex_doc_captions module**

```python
class ipypublish.preprocessors.latex_doc_captions.LatexCaptions(**kw):
    Bases: nbconvert.preprocessors.base.Preprocessor
    a preprocessor to:
      1. find cells with a ipub.caption meta-tag, extract the caption and label to a dict and remove the cell
      2. find cells with the found labels and replace their captions

    add_prefix
      A boolean (True, False) trait.
```

30 Chapter 2. Badges
**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.

---

**ipypublish.preprocessors.latex_doc_defaults** module

*class* ipypublish.preprocessors.latex_doc_defaults.MetaDefaults(**kw**)

**Bases:** nbconvert.preprocessors.base.Preprocessor

A preprocessor which enters default metadata tags into all cell metadata, without overriding any currently set

**cell_defaults**

An instance of a Python dict.

**nb_defaults**

An instance of a Python dict.

**overwrite**

A boolean (True, False) trait.

**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.

---

**ipypublish.preprocessors.latex_doc_defaults**.flatten *(d, key_as_tuple=True, sep='*')*

Get nested dict as `{key:val,...}`, where key is tuple/string of all nested keys

**Parameters**

- **d** *(dict)* –
- **key_as_tuple** *(bool)* – whether keys are list of nested keys or delimited string of nested keys
- **sep** *(str)* – if key_as_tuple=False, delimiter for keys
Examples

```python
>>> from pprint import pprint

>>> d = {1:{"a":"A"},2:{"b":"B"}}
>>> pprint(flatten(d))
{(1, 'a'): 'A', (2, 'b'): 'B'}

>>> d = {1:{"a":"A"},2:{"b":"B"}}
>>> pprint(flatten(d,key_as_tuple=False))
{'1.a': 'A', '2.b': 'B'}
```

**ipypublish.preprocessors.latex_doc_html module**

```python
class ipypublish.preprocessors.latex_doc_html.LatexDocHTML(*args, **kwargs)
Bases: nbconvert.preprocessors.base.Preprocessor
processing of ipub metatags, specific to html
  • import embedded html files
  • add refmap key to references for {label:reference name} lookup e.g. {"fig:test":"fig. 1"}
  • add caption_prefix tag for floats with correct numbering/name e.g. cell.metadata.ipub.figure.caption_prefix
    = "Figure 1: 
embed_html (cell, path)
  a new cell, based on embedded html file
filesfolder
  A trait for unicode strings.
metapath
  A trait for unicode strings.
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.
resolve_path (fpath, filepath)
  resolve a relative path, w.r.t. another filepath
src_name
  A trait for unicode strings.
```
ipypublish.preprocessors.latex_doc_links module

class ipypublish.preprocessors.latex_doc_links.LatexDocLinks(**kw)

Bases: nbconvert.preprocessors.base.Preprocessor

a preprocessor to resolve file paths in the ipub metadata section

retrieve external file paths from metadata, resolve where they are, if the path is relative make sure that the link
points to a single folder add ‘external_file_paths’ and ‘bibliopath’ (if present) to resources

filesfolder
    A trait for unicode strings.

metapath
    A trait for unicode strings.

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.

resolve_path (fpath, filepath)
    resolve a relative path, w.r.t. another filepath

ipypublish.preprocessors.latextags_to_html module

class ipypublish.preprocessors.latextags_to_html.DefaultFormatter (default=“)

Bases: string.Formatter

get_value (key, args, kwds)

class ipypublish.preprocessors.latextags_to_html.LatexTagsToHTML(*args,

**kwargs)

Bases: nbconvert.preprocessors.base.Preprocessor

a preprocessor to find latex tags (like cite{abc} or todo[color]{stuff}) and:

1. attempt to process them into a html friendly format
2. remove them entirely if this is not possible

for ref or cref, attempts to use resources.refmap to map labels to reference names for labels not found in re-

sources.refmap the reference name is ‘<name> <number>’, where: - <name> is either ref or, if labelbycolon
is True and the label has a colon, all text before the colon - <number> iterate by order of first appearance of a
particular label

NB: should be applied after LatexDocHTML, if you want resources.refmap to be available
Examples

```python
>>> from nbformat import NotebookNode
>>> from jsonextended.utils import MockPath

>>> processor = LatexTagsToHTML()

>>> bibfile = MockPath(is_file=True, content='''
... @article{bibkey,
...   title = {the title},
...   doi = {10.1134/S0018143916050209},
...   author = {Surname, A. Name},
...   date = {2016-09-01},
... }
... ''')

>>> resources = NotebookNode({'

>>> cell = NotebookNode({

>>> nb = NotebookNode({

>>> print(nb.cells[0].source)

>>> cell.source = "\\unknown\{test\}"

>>> nb, _ = processor.preprocess(nb, resources)

>>> print(nb.cells[0].source)

>>> cell.source = "\\ref\{label\}\\unknown\{test\}"

>>> nb, _ = processor.preprocess(nb, resources)

>>> print(nb.cells[0].source)

>>> cell.source = "\\label\{test\}"

>>> nb, _ = processor.preprocess(nb, resources)

>>> print(nb.cells[0].source)

>>> cell.source = "\\cite\{bibkey\}"

>>> nb, _ = processor.preprocess(nb, resources)

>>> print(nb.cells[0].source)

>>> cell.source = "\\begin\{equation\}x=a+b\\end\{equation\}"

>>> nb, _ = processor.preprocess(nb, resources)

>>> print(nb.cells[0].source)

```

bibformat

A trait for unicode strings.
convert (source, resources)

convert a a string with tags in

Example

```python
>>> source = r'''
... References to \cref{fig:example}, \cref{tbl:example}, \cref{eqn:example_sympy} and \cref{code:example_mpl}.
... Referencing multiple items: \cref{fig:example,fig:example_h,fig:example_v}.
... An unknown latex tag.\unknown{zelenyak_molecular_2016}
...''
>>> processor = LatexTagsToHTML()
>>> print(processor.convert(source,{}))
References to <a href="{id_home_prefix}fig:example">fig. 1</a>, <a href="{id_home_prefix}tbl:example">tbl. 1</a>, <a href="{id_home_prefix}eqn:example_sympy">eqn. 1</a> and <a href="{id_home_prefix}code:example_mpl">code. 1</a>.
Referencing multiple items: <a href="{id_home_prefix}fig:example">fig. 1</a>, <a href="{id_home_prefix}fig:example_h">fig. 2</a> and <a href="{id_home_prefix}fig:example_v">fig. 3</a>.
An unknown latex tag.
```

labelbycolon
A boolean (True, False) trait.

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.

process_bib_entry (entry)
work out the best way to represent the bib entry

static read_bibliography (path)
read a bibliography

regex
A trait for unicode strings.

replace_reflabel (name, resources)
find a suitable html replacement for a reference label
the links are left with a format hook in them: {id_home_prefix}, so that an nbconvert filter can later replace it this is particularly useful for slides, which require a prefix #/<slide_number><label>

2.9. Package API 35
Example text from the provided image:

```
    rreplace(source, target, replacement, replacements=1)
    replace in string, from right-to-left

ipypublish.preprocessors.latextags_to_html.safe_str(obj)
```

### ipypublish.preprocessors.slides_from_markdown module

**class** ipypublish.preprocessors.slides_from_markdown.FinalCells(header_slide)

Bases: object

a class that stores cells

append(cell)

finalize()

first()

last()

mkdcell(source, metadata, slidetype)

**class** ipypublish.preprocessors.slides_from_markdown.MarkdownSlides(**kw)

Bases: nbconvert.preprocessors.base.Preprocessor

a preprocessor to setup the notebook as an ipyslideshow, according to a set of rules

- markdown cells containiing # headers are broken into individual cells
- any cells where ipub.ignore=True is set to 'skip'
- any code cells with no other ipub tags are set to 'skip'
- any header level >= column_level starts a new column
- else, any header level >= row_level starts a new row
- if max_cells is not 0, then breaks to a new row after <max_cells> cells

**autonumbering**

A boolean (True, False) trait.

**column_level**

An int trait.

**header_slide**

A boolean (True, False) trait.

**max_cells**

An int trait.

**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.

36 Chapter 2. Badges
**row_level**
An int trait.

```python
>>> header_level('# title')
1
>>> header_level('### title')
3
```

**Examples**

```python
>>> is_header("abc",0)
False
>>> is_header("#",0)
False
>>> is_header("# title",0)
True
>>> is_header("### title",3)
True
>>> is_header("### title",2)
False
```

**Examples**

```python
>>> number_title("# title",[])
('# 1. title', [1])
>>> number_title("## title",[])
('# 1.1. title', [1, 1])
>>> number_title("# title",[1,1])
('# 2. title', [2])
>>> number_title("## title",[2,1])
('# 2.2. title', [2, 2])
>>> number_title("### title a#bc",[2])
('#### 2.1.1. title a#bc', [2, 1, 1])
>>> number_title("### title a#bc",[2,1,2,3])
('#### 2.1.3. title a#bc', [2, 1, 3])
```

**Examples**

```python
>>> header_level('# title')
1
>>> header_level('### title')
3
```

**Examples**

```python
>>> is_header("abc",0)
False
>>> is_header("#",0)
False
>>> is_header("# title",0)
True
>>> is_header("### title",3)
True
>>> is_header("### title",2)
False
```

**Examples**

```python
>>> number_title("# title",[])
('# 1. title', [1])
>>> number_title("## title",[])
('# 1.1. title', [1, 1])
>>> number_title("# title",[1,1])
('# 2. title', [2])
>>> number_title("## title",[2,1])
('# 2.2. title', [2, 2])
>>> number_title("### title a#bc",[2])
('#### 2.1.1. title a#bc', [2, 1, 1])
>>> number_title("### title a#bc",[2,1,2,3])
('#### 2.1.3. title a#bc', [2, 1, 3])
```
**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.

**split**
A boolean (True, False) trait.

```python
ipypublish.preprocessors.split_outputs.merge(a, b, path=None, overwrite=True)
```
merges b into a

**Examples**

```python
>>> from pprint import pprint
>>> pprint(merge({'a':{ 'b':1}, 'c':3},{'a':{ 'b':2}}))
{'a': {'b': 2}, 'c': 3}
```

**Module contents**

**ipypublish.scripts package**

**Submodules**

**ipypublish.scripts.ipynb_latex_setup module**

Some setup for improved latex/pdf output at top of workbook, use:

```python
from ipynb_latex_setup import *
```

**create_test_image** *(size=(50, 50))*

**images_gridconcat** *(pathlist, width=700, height=700, aspaths=True, hgap=0, vgap=0)*

concatenate multiple images in a grid

**Parameters**
- **pathlist** *(list[list])* – if aspaths=True, list of path strings, else list of PIL.Image instances each sub list constitutes a row
- **width** *(int)* – maximum width of final image
- **height** *(int)* – maximum height of final image
- **hgap** *(int)* – size of horizontal space between images
• **vgap** *(int)* – size of vertical space between images

**Returns** image

**Return type** PIL.Image

### ipypublish.scripts.ipynb_latex_setup.images_hconcat *(images, width=700, height=700, gap=0, aspaths=True)*

concatenate multiple images horizontally

**Parameters**

- **images** *(list)* – if aspaths=True, list of path strings, else list of PIL.Image instances
- **width** *(int or list[int])* – maximum width of final image, or of individual images
- **height** *(int or list[int])* – maximum height of final image, or of individual images
- **gap** *(int)* – size of space between images

**Returns** image

**Return type** PIL.Image

### Examples

```python
>>> img_path = create_test_image(size=(50,50))
>>> img = images_hconcat([img_path, img_path])
>>> img.size
(100, 50)
```

```python
>>> img_path = create_test_image(size=(50,50))
>>> img = images_hconcat([img_path, img_path], width=40, height=40)
>>> img.size
(40, 20)
```

```python
>>> img_path = create_test_image(size=(50,50))
>>> img = images_hconcat([img_path, img_path], width=[40,30])
>>> img.size
(70, 40)
```

```python
>>> img_path = create_test_image(size=(50,50))
>>> img = images_hconcat([img_path, img_path], gap=10)
>>> img.size
(110, 50)
```

### ipypublish.scripts.ipynb_latex_setup.images_read *(paths)*

read a list of image paths to a list of PIL.IMAGE instances

### ipypublish.scripts.ipynb_latex_setup.images_vconcat *(images, width=700, height=700, gap=0, aspaths=True)*

concatenate multiple images vertically

**Parameters**

- **images** *(list)* – if aspaths=True, list of path strings, else list of PIL.Image instances
- **width** *(int or list[int])* – maximum width of final image, or of individual images
• **height** (*int* or *list[int]*) – maximum height of final image, or of individual images

• **gap** (*int*) – size of space between images

Returns image

Return type PIL.Image

**Examples**

```python
>>> img_path = create_test_image(size=(50,50))
>>> img = images_vconcat([img_path, img_path])
>>> img.size
(50, 100)
```

```python
>>> img_path = create_test_image(size=(50,50))
>>> img = images_vconcat([img_path, img_path], width=40, height=40)
>>> img.size
(20, 40)
```

```python
>>> img_path = create_test_image(size=(50,50))
>>> img = images_vconcat([img_path, img_path], width=[40, 30])
>>> img.size
(40, 70)
```

```python
>>> img_path = create_test_image(size=(50,50))
>>> img = images_vconcat([img_path, img_path], gap=10)
>>> img.size
(50, 110)
```

**ipypublish.scripts.nbmerge module**

**usage:**

```bash
python nbmerge.py directory_name > merged.ipynb
```

ipypublish.scripts.nbmerge.alphanumeric_sort(*l*)

sort key.name alphabetically

**Parameters**

• *list[str]*

ipypublish.scripts.nbmerge.merge_notebooks(*ipynb_path*, *ignore_prefix*='__', *to_str*=False, *as_version*=4)

merge one or more ipynb’s, if more than one, then the meta data is taken from the first

**Parameters**

• *ipynb_path* (*str* or *pathlib.Path*)

• *ignore_prefix* (*str*) – ignore filename starting with this prefix

• *to_str* (*bool*) – return as a string, else return nbformat object

• *as_version* (*int*) – notebook format version

**Returns**

• *finalnb* (*jupyter.notebook*)
• **meta_path** *(pathlib.Path)* – path to notebook containing meta file

**ipypublish.scripts.pdfexport module**

a module for exporting latex file to pdf TODO could this be refactored as nbconvert postprocessor

```python
class ipypublish.scripts.pdfexport.change_dir(new_path)
    Bases: object

    Context manager for changing the current working directory
```

```python
class ipypublish.scripts.pdfexport.export_pdf(texpath, outdir, files_path=None, convert_in_temp=False, html_viewer=True, debug_mode=False)
```

```python
class ipypublish.scripts.pdfexport.run_conversion(texpath, out_folder, files_folder=None, debug_mode=False)
```

run latexmk conversion

**ipypublish.scripts.reveal_serve module**

serve HTML page TODO the RevealServer setting should be available at front end

```python
class ipypublish.scripts.reveal_serve.ProxyHandler(application, request, **kwargs)
    Bases: tornado.web.RequestHandler

    handler the proxies requests from a local prefix to a CDN

    def finish_get(self, response)
        finish the request

    get(self, prefix, url)
        proxy a request to a CDN
```

```python
class ipypublish.scripts.reveal_serve.RevealServer(**kwargs)
    Bases: traitlets.config.configurable.LoggingConfigurable

    Post processor designed to serve files

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

```python
ip
    A trait for unicode strings.

open_in_browser
    A boolean (True, False) trait.

port
    An int trait.

reveal_cdn
    A trait for unicode strings.

reveal_prefix
    A trait for unicode strings.

serve(input)
    Serve the build directory with a webserver.
Module contents

**ipypublish.templates package**

**Subpackages**

**ipypublish.templates.outline_schemas package**

**Submodules**

**ipypublish.templates.outline_schemas.convert_format_str module**

**Module contents**

contains Jinja outline template schema

**ipypublish.templates.segments package**

**Module contents**

contains Jinja outline segments

**Submodules**

**ipypublish.templates.create_template module**

create template

philosophy is only turn stuff on when we want


ipypublish.templates.create_template.create_template(outline_template, outline_name, segment_datas, outpath=None)

build a latex jinja template from;

- a jinja(2) template outline, which may contain segment placeholders,
- and json segment files adhering to the segment.schema.json schema

if a segment contains the key “overwrite”, then its value should be a list of keys, such that these key values overwrite any entries before

**Parameters**

- **outline_template**(str)–
- **segment_datas**(tuple or dict)–
- **outpath**(None or str)–if not None, output to path
**ipypublish.templates.create_template.multireplace**(string, replacements)

Given a string and a replacement map, it returns the replaced string.

From https://gist.github.com/bgusach/a967e0587d6e01e889fd1d776c5f3729

**Parameters**

- **string** *(str)* – string to execute replacements on
- **replacements** *(dict)* – replacement dictionary {find value: replacement}

**Return type** *str*

**Module contents**

**Submodules**

**ipypublish.utils module**

**ipypublish.utils.get_module_path**(module)

return a directory path to a module

**ipypublish.utils.handle_error**(msg, err_type, logger, raise_msg=None, log_msg=None)

handle an error, by logging it, then raising an exception

**ipypublish.utils.read_file_from_directory**(dir_path, file_name, jtype, logger, as_json=False)

load a file situated in a directory

if as_json=True load file contents to a dict

**ipypublish.utils.read_file_from_module**(module_path, file_name, jtype, logger, as_json=False)

load a file situated in a python module

if as_json=True load file contents to a dict

**Module contents**

**2.10 Releases**

**2.10.1 v0.8.0 - Outline templates now use a jinja file, instead of json**

- extracted templates into separate files
- use template outline file instead of json schema
- improve front end logging
- update documentation
- version bump
- added tests

**2.10.2 v0.7.1 - Improved the dict_to_kwds filter and added biboptions metatag**

See Bibliography Meta-data
2.10.3 v0.7.0 - Major API Update

- Converted export configurations and templates from python to JSON
- Added validation schema for configurations and templates
- added option to control style of bibliography in latex
- Converted script executables to console entry points
- Updated test configuration from nose to pytest
  - added many more tests for all export configurations and user interface
  - fixed Mac Os build on Travis
- Improved user interface
- Added ipynb to python file (with commented metadata) exporter
- Updated documentation with new API and how to convert plugins

2.10.4 v0.6.8 - version bump to initiate Zenodo

2.10.5 v0.6.7 - Added support for raw cells

Raw output is now included in the latex (if raw format is latex), and html (if raw format is html)

2.10.6 v0.6.5 - Minor Update

2.10.7 v0.6.4 - Encoding Bug Fixes for Python < 3.6

and addition of documentation

2.10.8 v0.6.3 - better support for latex math environments

2.10.9 v0.6.2 - Améliorations!

- added language translation
- added width/height options for latex figures
- changed embedded html to be iframes, with lazy loading for reveal slides
- added titles and author for html and slides
- bibtxparser uses “link” rather than “url” key (fixed)
- fixed regex for headers (one or more # not zero or more)
- allow codecells with no outputs
- added ansi colors for latex listings
- added adjust box for resizing tables too wide to fit in page width
2.10.10 v0.6.1 - added output level metadata

see https://github.com/chrisjsewell/ipypublish#metadata-tags

2.10.11 v0.6.0 - changed top-level meta tag from latex_doc -> ipub

To reflect that it also applys to html/slides output
also:
- improved control of slide output
- changed from using utf8x -> xelatex, for handling font encoding
- added mdown output tag

2.10.12 v0.5.3 - Small bug fix for html caption prefixing

- moved html caption prefixing to LatexCaption, so that captions from other cells are prefixed

2.10.13 v0.5.2 - Slide autonumbering and captions from code output

2.10.14 v0.5.1 - Improvements to Slide Output and Smart Slide Creation

slide rows/columns partitioned by markdown headers
also improved latex listings default options for text & stream data

2.10.15 v0.5.0 - Default Conversion Plugins & Enhancements to HTML/Slides Conversion

- added auto numbering and correct reference hyperlinks for figures/tables/equations/code in html & slides
- added text meta-tag, default meta-tag post processor, and additional converters based on it
- added embeddable html

2.10.16 v0.4.1 - added universal bdist flag

2.10.17 v0.4.0 - Introducing nbpresent: for reveal.js slideshow creation and serving

a lot of refactoring of html template creation
improvement of command line argument processing
introduction of preprocessors
general awesomeness
2.10.18 v0.3.0 - First full, tested pypi release!

2.10.19 v0.2 - New Latex Metadata convention

Now all under “latex_doc” key with no “latex_” prefix, e.g.

```
{
  "latex_doc": {
    "ignore": true
  }
}
```

instead of:

```
{"latex_ignore": true}
```

2.10.20 v0.1 - An initial release

before changing latex meta tag convention

2.11 Example Export Configuration

```
{
  "description": [  
    "latex article in the main ipypublish format, preprocessed with default metadata, -
    "tags:",
    "- all output/code/error is rendered",
    "- a basic titlepage and table of contents and tables of figures/tables/code",
  ],
  "exporter": {
    "class": "nbconvert.exporters.LatexExporter",
    "filters": {
      "remove_dollars": "ipypublish.filters.filters.remove_dollars",
      "first_para": "ipypublish.filters.filters.first_para",
      "create_key": "ipypublish.filters.filters.create_key",
      "dict_to_kwds": "ipypublish.filters.filters.dict_to_kwds",
      "ansi2listings": "ipypublish.filters.ansi_listings.ansi2listings",
      "is_equation": "ipypublish.filters.filters.is_equation",
      "strip_ext": "ipypublish.filters.filters.strip_ext"
    },
    "preprocessors": [  {
      "class": "ipypublish.preprocessors.latex_doc_defaults.MetaDefaults",
      "args": {
        "cell_defaults": {
```
(continues on next page)
2.11. Example Export Configuration
2.12 Export Configuration Schema

```json
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "description": "a configuration for exporting an ipynb in ipypublish",
  "type": "object",
  "additionalProperties": true,
  "required": [
    "description",
    "exporter",
    "template"
  ],
  "properties": {
    "$schema": {
      "type": "string"
    },
    "description": {
      "description": "a description of the configuration",
      "type": "array",
      "minItems": 1,
      "items": {
        "type": "string"
      }
    },
    "exporter": {
      "description": "an exporter to be used by nbconvert",
      "type": "string"
    }
  }
}
```
"type": "object",
"additionalProperties": false,
"required": [
  "class",
  "filters",
  "preprocessors"
],
"properties": {
  "class": {
    "description": "the python class path for the exporter",
    "type": "string"
  },
  "filters": {
    "description": "the filter functions to be used by the exporter",
    "type": "object",
    "patternProperties": {
      ".+": {
        "type": "string"
      }
    }
  },
  "preprocessors": {
    "description": "the preprocessor classes to be used by the exporter",
    "type": "array",
    "items": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "class",
        "args"
      ],
      "properties": {
        "class": {
          "description": "the python class path for the preprocessor",
          "type": "string"
        },
        "args": {
          "description": "arguments taken by the preprocessor",
          "type": "object"
        }
      }
    }
  }
},
"other_args": {
  "description": "additional arguments to be fed to the traitlets Config",
  "type": "object"
},
"template": {
  "description": "configuration to build a Jinja template",
  "type": [
    "null",
    "object"
  ]
}
ipypublish Documentation, Release 0.8.0

(continued from previous page)

],
  "additionalProperties": false,
  "required": [
    "outline"
  ],
  "properties": {
    "outline": {
      "description": "the outline schema, containing the jinja template and schema for segments",
      "$ref": "#/definitions/json_location"
    },
    "segments": {
      "description": "segments to be inserted into the outline",
      "type": "array",
      "items": {
        "$ref": "#/definitions/json_location"
      }
    }
  }
},
"definitions": {
  "json_location": {
    "oneOf": [
      {
        "type": "object",
        "additionalProperties": false,
        "required": [
          "module",
          "file"
        ],
        "properties": {
          "module": {
            "description": "the python module that the json file resides in",
            "type": "string"
          },
          "file": {
            "description": "the filename of the json file",
            "type": "string"
          }
        }
      },
      {
        "type": "object",
        "additionalProperties": false,
        "required": [
          "directory",
          "file"
        ],
        "properties": {
          "directory": {
            "description": "the folder path that the json file resides in",
            "type": "string"
          },
          "file": {
            "type": "string"
          }
        }
      }
    ]
  }
}
(continues on next page)
2.13 Template Outline Example

```{template}
{%- extends 'display_priority.tpl' -%}

@ipubreplace{below}{globals}

{% HTML Setup
  %
  HTML ================
  
  {% block header %}
  An html document created by ipypublish @ipubreplace{below}{ipypub_version}
  @ipubreplace{below}{meta_docstring}
  -->
  
  <!DOCTYPE html>
  <html>
  <head>
    <meta charset="utf-8" />
    {% block html_head %}
    @ipubreplace{below}{html_header}
    {% endblock html_head %}
  </head>
  {% endblock header %}
  
  {% block body %}
  <body>
  @ipubreplace{below}{html_body_start}
  {{ super() }}
  @ipubreplace{below}{html_body_end}
  </body>
  {% endblock body %}
  
  {% block footer %}
  @ipubreplace{below}{html_footer}
  {% endblock footer %}
  
  % Notebook Input
  %
  
  {% block any_cell scoped %}
  @ipubreplace{below}{notebook_all}
  {% endblock any_cell %}
```

2.13. Template Outline Example
{% block input_group %}@ipubreplace{above}{notebook_input_code_pre}
// super() //
@ipubreplace{below}{notebook_input_code_post}
{% endblock input_group %}

{% block in_prompt %}@ipubreplace{below}{notebook_input_code_prompt}
{% endblock in_prompt %}

{% block input scoped %}@ipubreplace{below}{notebook_input_code}
{% endblock input %}

{% block rawcell scoped %}@ipubreplace{above}{notebook_input_raw_pre}
@ipubreplace{below}{notebook_input_raw}
@ipubreplace{below}{notebook_input_raw_post}
{% endblock rawcell %}

{% block markdowncell scoped %}@ipubreplace{above}{notebook_input_markdown_pre}
@ipubreplace{below}{notebook_input_markdown}
@ipubreplace{below}{notebook_input_markdown_post}
{% endblock markdowncell %}

{% block unknowncell scoped %}@ipubreplace{above}{notebook_input_unknown_pre}
@ipubreplace{below}{notebook_input_unknown}
@ipubreplace{below}{notebook_input_unknown_post}
{% endblock unknowncell %}

%% Notebook Outbook
%% ================

{% block output %}@ipubreplace{above}{notebook_output_pre}
@ipubreplace{below}{notebook_output_prompt}
// super() //
@ipubreplace{below}{notebook_output_post}
{% endblock output %}

% Redirect execute_result to display data priority.
{% set extra_class="output_execute_result" %}
{% block data_priority %}
@ipubreplace{below}{notebook_output}
{% endblock %}

{% set extra_class="" %}
{% endblock execute_result %}

{% block error %}@ipubreplace{above}{notebook_output_error_pre}
@ipubreplace{below}{notebook_output_error}
@ipubreplace{below}{notebook_output_error_post}
{% endblock error %}

2.13. Template Outline Example
2.14 Example of a Template Segment

```
{ "$schema": ".../segment.schema.json", 
"identifier": "ipypublish-biblio_natbib", 
"description": "with the main ipypublish bibliography", 
"segments": { 
  "document_packages": [ 
    "", 
    "% bibliography formatting", 
    "((%- if nb.metadata.ipub and nb.metadata.ipub.biboptions: *)\n% bibliography formatting 
    "\"usepackage\{{\{{nb.metadata.ipub.biboptions | dict_to_kwds})\}\}\{natbib\}"", 
    "((%- else *))", 
    "\"usepackage\{{numbers, square, super, sort&compress}\}\{natbib\}"", 
    "((%- endif *))")", 
    "% hyperlink doi's", 
    "\"usepackage\{doi\}"
  ],
  "document_bibliography": [ 
    "", 
    "((%- if nb.metadata.ipub: -*))",
  ]
}
```
"\bibliographystyle{unsrtnat}",
"\bibliography{{{{ filename }}}},

\"]
\}
CHAPTER 3

Index and Search

- genindex
- modindex
- search
ipypublish, 43
ipypublish.convert, 25
ipypublish.convert.config_manager, 24
ipypublish.convert.main, 24
ipypublish.export_plugins, 25
ipypublish.filters, 27
ipypublish.filters.ansi_listings, 25
ipypublish.filters.filters, 26
ipypublish.filters.meta_to_yaml, 27
ipypublish.filters.replace_string, 27
ipypublish.frontend, 29
ipypublish.frontend.nbpresent, 27
ipypublish.frontend.nbpublish, 28
ipypublish.frontend.shared, 28
ipypublish.port_api, 30
ipypublish.port_api.plugin_to_json, 29
ipypublish.port_api.tpl_dct_to_json, 30
ipypublish.preprocessors, 38
ipypublish.preprocessors.crop_cells, 30
ipypublish.preprocessors.latex_doc_captions, 30
ipypublish.preprocessors.latex_doc_defaults, 31
ipypublish.preprocessors.latex_doc_html, 32
ipypublish.preprocessors.latex_doc_links, 33
ipypublish.preprocessors.latextags_to_html, 33
ipypublish.preprocessors.slides_from_markdown, 36
ipypublish.preprocessors.split_outputs, 37
ipypublish.scripts, 42
ipypublish.scripts.ipynb_latex_setup, 38
ipypublish.scripts.nbmerge, 40
ipypublish.scripts.pdfexport, 41
ipypublish.scripts.reveal_serve, 41
ipypublish.templates, 43
ipypublish.templates.create_template, 42
ipypublish.templates.outline_schemas, 42
ipypublish.templates.segments, 42
ipypublish.utils, 43
Index

A
add_prefix (ipypublish.preprocessors.latex_doc_caption.LatexDocCaption.attribute), 30
alphanumeric_sort() (in module ipypublish.scripts.nbmerge), 40
ansi2listings() (in module ipypublish.filters.ansi_listings), 25
append() (ipypublish.preprocessors.slides_from_markdown.FinalCells method), 36
assess_syntax() (in module ipypublish.port_api.plugin_to_json), 29
assess_syntax() (in module ipypublish.port_api.tpl_dct_to_json), 30
ast_to_json() (in module ipypublish.port_api.plugin_to_json), 30
autonumbering (ipypublish.preprocessors.slides_from_markdown.MarkdownSlides attribute), 36

B
bibformat (ipypublish.preprocessors.latextags_to_html.LatexTagsToHTML.attribute), 34

C
cell_defaults (ipypublish.preprocessors.latex_doc_defaults.MetaDefaults.attribute), 31
change_dir (class in ipypublish.scripts.pdfexport), 41
column_level (ipypublish.preprocessors.slides_from_markdown.MarkdownSlides attribute), 36
convert() (ipypublish.preprocessors.latextags_to_html.LatexTagsToHTML.method), 35
convert_config() (in module ipypublish.convert.main), 24
convert_exporter_cls() (in module ipypublish.convert.config_manager), 24
convert_json() (in module ipypublish.port_api.plugin_to_json), 29
crop_key() (in module ipypublish.filters.filters), 26
crop_template() (in module ipypublish.port_api.tpl_dct_to_json), 30
de_default_formatter (class in ipypublish.preprocessors.crop_cells), 30
dict_to_config() (in module ipypublish.convert.main), 24
dict_to_kwds() (in module ipypublish.filters.filters), 26
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
de_format() (in module ipypublish.preprocessors.latextags_to_html.LatexDocHTML.method), 32
debug (ipypublish.preprocessors.crop_cells.CropCells attribute), 30
derror() (ipypublish.frontend.shared.CustomParser method), 28
dict_to_json() (in module ipypublish.convert.main), 24
dict_to_json() (in module ipypublish.filters.filters), 26
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
export_notebook() (in module ipypublish.convert.main), 24
export_pdf() (in module ipypublish.scripts.pdfexport), 41

D
default_formatter (class in ipypublish.preprocessors.latextags_to_html), 33
dict_to_config() (in module ipypublish.convert.main), 24
dict_to_kwds() (in module ipypublish.filters.filters), 26
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29
dict_to_json() (in module ipypublish.port_api.tpl_dct_to_json), 30
dict_to_json() (in module ipypublish.port_api.plugin_to_json), 29

E
eembed_html() (ipypublish.preprocessors.latex_doc.html.LatexDocHTML.method), 32
eexport_notebook() (in module ipypublish.convert.main), 24

F
files_folder (ipypublish.preprocessors.latex_doc_links.LatexDocLinks.attribute), 32
files_folder (ipypublish.preprocessors.latex_doc_links.LatexDocLinks.attribute), 33
**Index**

```
FinalCells (class in ipypublish.preprocessors.slides_from_markdown), 36
finalize() (ipypublish.preprocessors.slides_from_markdown method), 36
finish_get() (ipypublish.scripts.reveal_serve.ProxyHandler method), 41
first() (ipypublish.preprocessors.slides_from_markdown.FinalCells method), 36
first_para() (in module ipypublish.filters.filters), 26
flatten() (in module ipypublish.preprocessors.latex_doc_defaults), 31
get() (ipypublish.scripts.reveal_serve.ProxyHandler method), 41
get_export_config_path() (in module ipypublish.convert.config_manager), 24
get_export_extension() (in module ipypublish.convert.config_manager), 24
get_module_path() (in module ipypublish.utils), 43
get_parser() (in module ipypublish.frontend.shared), 28
get_plugin_str() (in module ipypublish.frontend.shared), 28
get_value() (ipypublish.preprocessors.latextags_to_html.DefaultFormatter method), 33
handle_error() (in module ipypublish.utils), 43
header_level() (in module ipypublish.preprocessors.slides_from_markdown), 37
header_slide (ipypublish.preprocessors.slides_from_markdown attribute), 36
images_gridconcat() (in module ipypublish.scripts.ipynb_latex_setup), 38
images_hconcat() (in module ipypublish.scripts.ipynb_latex_setup), 39
images_read() (in module ipypublish.scripts.ipynb_latex_setup), 39
images_vconcat() (in module ipypublish.scripts.ipynb_latex_setup), 39
ip (ipypublish.scripts.reveal_serve.RevealServer attribute), 41
ipypublish (module), 43
ipypublish.convert (module), 25
ipypublish.convert.config_manager (module), 24
ipypublish.convert.main (module), 24
ipypublish.export_plugins (module), 25
ipypublish.filters (module), 27
ipypublish.filters.ansi_listings (module), 25
ipypublish.filters.filters (module), 26
ipypublish.filters.meta_to_yaml (module), 27
ipypublish.filters.replace_string (module), 27
ipypublish.frontend (module), 29
ipypublish.frontend.nbpresent (module), 27
ipypublish.frontend.npbundle (module), 28
ipypublish.frontend.shared (module), 28
ipypublish.port_api (module), 30
ipypublish.port_api.plugin_to_json (module), 29
ipypublish.port_api.tpl_dct_to_json (module), 30
ipypublish.preprocessors (module), 38
ipypublish.preprocessors.crop_cells (module), 30
ipypublish.preprocessors.latex_doc_defaults (module), 30
ipypublish.preprocessors.latex_doc_caption (module), 30
ipypublish.preprocessors.latex_doc_defaults (module), 31
ipypublish.preprocessors.latex_doc_html (module), 32
ipypublish.preprocessors.latex_doc_links (module), 33
ipypublish.preprocessors.latextags_to_html (module), 33
ipypublish.preprocessors.slides_from_markdown (module), 36
ipypublish.preprocessors.split_outputs (module), 37
ipypublish.scripts (module), 42
ipypublish.scripts.ipynb_latex_setup (module), 38
ipypublish.scripts.nbmerge (module), 40
ipypublish.scripts.pdfexport (module), 41
ipypublish.scripts.reveal_serve (module), 41
ipypublish.templates (module), 43
ipypublish.templates.create_template (module), 42
ipypublish.templates.outline_schemas (module), 42
ipypublish.templates.segments (module), 42
ipypublish.utils (module), 43
is_equation() (in module ipypublish.filters.filters), 26
is_header() (in module ipypublish.preprocessors.slides_from_markdown), 37
iter_all_export_infos() (in module ipypublish.convert.config_manager), 24
iter_all_export_paths() (in module ipypublish.convert.config_manager), 24
labelbycolon (ipypublish.preprocessors.latextags_to_html.LatexTagsToHTML attribute), 35
last() (ipypublish.preprocessors.slides_from_markdown.FinalCells method), 35
LatexCaptions (class in ipypublish.preprocessors.slides_from_markdown.FinalCells attribute), 36
LatexDocHTML (class in ipypublish.preprocessors.slides_from_markdown.FinalCells method), 36
LatexDocLinks (class in ipypublish.preprocessors.slides_from_markdown.FinalCells method), 36
LatexTagsToHTML (class in ipypublish.preprocessors.slides_from_markdown.FinalCells method), 36
```
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Module/Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>load_export_config()</td>
<td>ipypublish.convert.config_manager</td>
</tr>
<tr>
<td>load_template()</td>
<td>ipypublish.convert.config_manager</td>
</tr>
<tr>
<td>MarkdownSlides</td>
<td>ipypublish.preprocessors.slides_from_markdown</td>
</tr>
<tr>
<td>max_cells</td>
<td>ipypublish.preprocessors.slides_from_markdown</td>
</tr>
<tr>
<td>merge()</td>
<td>ipypublish.preprocessors.split_outputs</td>
</tr>
<tr>
<td>merge_notebooks()</td>
<td>ipypublish.scripts.nbmerge</td>
</tr>
<tr>
<td>meta_to_yaml()</td>
<td>ipypublish.filters.meta_to_yaml</td>
</tr>
<tr>
<td>MetaDefaults</td>
<td>ipypublish.preprocessors.latex_doc_defaults</td>
</tr>
<tr>
<td>metapath</td>
<td>ipypublish.preprocessors.latex_doc_html.LatexDocHTML</td>
</tr>
<tr>
<td>mkdcell()</td>
<td>ipypublish.preprocessors.slides_from_markdown</td>
</tr>
<tr>
<td>multireplace()</td>
<td>ipypublish.templates.create_template</td>
</tr>
<tr>
<td>nb_defaults</td>
<td>ipypublish.preprocessors.latex_doc_defaults</td>
</tr>
<tr>
<td>nbpresent()</td>
<td>ipypublish.frontend.nbpresent</td>
</tr>
<tr>
<td>nbpublish()</td>
<td>ipypublish.frontend.nbpublish</td>
</tr>
<tr>
<td>number_title()</td>
<td>ipypublish.preprocessors.slides_from_markdown</td>
</tr>
<tr>
<td>open_in_browser</td>
<td>ipypublish.scripts.reveal_serve.RevealServer</td>
</tr>
<tr>
<td>overwrite</td>
<td>ipypublish.preprocessors.latex_doc_defaults.MetaDefaults</td>
</tr>
<tr>
<td>parse_options()</td>
<td>ipypublish.frontend.shared</td>
</tr>
<tr>
<td>port</td>
<td>ipypublish.scripts.reveal_serve.RevealServer</td>
</tr>
<tr>
<td>postprocess_nb()</td>
<td>ipypublish.convert.main</td>
</tr>
<tr>
<td>preprocess()</td>
<td>ipypublish.preprocessors.crop_cells.CropCells</td>
</tr>
<tr>
<td>preprocess</td>
<td>ipypublish.preprocessors.latex_doc_captions.LatexCaptions</td>
</tr>
<tr>
<td>preprocess</td>
<td>ipypublish.preprocessors.latex_doc_defaults.MetaDefaults</td>
</tr>
<tr>
<td>preprocess</td>
<td>ipypublish.preprocessors.latex_doc_html.LatexDocHTML</td>
</tr>
<tr>
<td>preprocess</td>
<td>ipypublish.preprocessors.latex_doc_links.LatexDocLinks</td>
</tr>
<tr>
<td>preprocess</td>
<td>ipypublish.preprocessors.latextags_to_html.LatexTagsToHTML</td>
</tr>
<tr>
<td>preprocess</td>
<td>ipypublish.preprocessors.slides_from_markdown.MarkdownSlides</td>
</tr>
<tr>
<td>preprocess</td>
<td>ipypublish.preprocessors.split_outputs.SplitOutputs</td>
</tr>
<tr>
<td>process_bib_entry()</td>
<td>ipypublish.preprocessors.latextags_to_html.LatexTagsToHTML</td>
</tr>
<tr>
<td>ProxyHandler</td>
<td>ipypublish.scripts.reveal_serve</td>
</tr>
<tr>
<td>read_bibliography()</td>
<td>ipypublish.preprocessors.latextags_to_html.LatexTagsToHTML</td>
</tr>
<tr>
<td>read_file_from_directory()</td>
<td>ipypublish.utils</td>
</tr>
<tr>
<td>recurse_convert()</td>
<td>ipypublish.filters.meta_to_yaml</td>
</tr>
<tr>
<td>regex</td>
<td>ipypublish.preprocessors.latextags_to_html.LatexTagsToHTML</td>
</tr>
<tr>
<td>remove_dollars()</td>
<td>ipypublish.filters.filters</td>
</tr>
<tr>
<td>replace_relabel()</td>
<td>ipypublish.preprocessors.latextags_to_html.LatexTagsToHTML</td>
</tr>
<tr>
<td>replace_string()</td>
<td>ipypublish.filters.replace_string</td>
</tr>
<tr>
<td>replace_template_path()</td>
<td>ipypublish.port_api.tpl_dct_to_json</td>
</tr>
<tr>
<td>resolve_path()</td>
<td>ipypublish.preprocessors.latex_doc_html.LatexDocHTML</td>
</tr>
<tr>
<td>resolve_path</td>
<td>ipypublish.preprocessors.latex_doc_links.LatexDocLinks</td>
</tr>
<tr>
<td>reveal_cdn</td>
<td>ipypublish.scripts.reveal_serve.RevealServer</td>
</tr>
<tr>
<td>reveal_prefix</td>
<td>ipypublish.scripts.reveal_serve.RevealServer</td>
</tr>
<tr>
<td>RevealServer</td>
<td>ipypublish.scripts.reveal_serve</td>
</tr>
</tbody>
</table>
row_level (ipy.publish.preprocessors.slides_from_markdown.MarkdownSlides attribute), 36
rreplace() (ipy.publish.preprocessors.latextags_to_html.LatexTagsToHTML method), 35
run() (in module ipy.publish.frontend.nbpresent), 27
run() (in module ipy.publish.frontend.nbpublish), 28
run_conversion() (in module ipy.publish.scripts.pdfexport), 41

S
safe_str() (in module ipy.publish.preprocessors.latextags_to_html), 36
serve() (ipy.publish.scripts.reveal_serve.RevealServer method), 41
split (ipy.publish.preprocessors.split_outputs.SplitOutputs attribute), 38
SplitOutputs (class in ipy.publish.preprocessors.split_outputs), 37
src_name (ipy.publish.preprocessors.latex_doc_html.LatexDocHTML attribute), 32
start (ipy.publish.preprocessors.crop_cells.CropCells attribute), 30
str_to_jinja() (in module ipy.publish.convert.config_manager), 24
strip_ext() (in module ipy.publish.filters.filters), 26

W
wrap_latex() (in module ipy.publish.filters.filters), 27
write_output() (in module ipy.publish.convert.main), 25