papermill Documentation

Release 2.7.0

nteract team

Mar 08, 2026

CONTENTS

1 Python Version Support **3**

2 Documentation **5**
- 2.1 Installation . . . 5
- 2.2 Usage . . . 5
- 2.3 Command Line Interface . . . 11
- 2.4 Extending papermill . . . 13
- 2.5 Troubleshooting . . . 19
- 2.6 Change Log . . . 19

3 API Reference **21**
- 3.1 Reference . . . 21

4 Indices and tables **43**

Python Module Index **45**

Index **47**

Papermill is a tool for parameterizing and executing Jupyter Notebooks.

Papermill lets you:

- **parameterize** notebooks
- **execute** notebooks

This opens up new opportunities for how notebooks can be used. For example:

- Perhaps you have a financial report that you wish to run with different values on the first or last day of a month or at the beginning or end of the year, **using parameters** makes this task easier.
- Do you want to run a notebook and depending on its results, choose a particular notebook to run next? You can now programmatically **execute a workflow** without having to copy and paste from notebook to notebook manually.

CHAPTER

ONE

PYTHON VERSION SUPPORT

This library currently supports python 3.10+ versions. As minor python versions are officially sunset by the python org papermill will similarly drop support in the future.

CHAPTER

TWO

DOCUMENTATION

These pages guide you through the installation and usage of papermill.

2.1 Installation

2.1.1 Installing papermill

From the command line:

```
python3 -m pip install papermill
```

2.1.2 Installing In-Notebook language bindings

In-Notebook language bindings provide helpers and utilities for using Papermill with a programming language.

Python bindings

No additional installation steps are required since python bindings are built into **papermill**.

2.2 Usage

For an interactive example that demonstrates the usage of papermill, click the Binder link below:

2.2.1 Using papermill

The general workflow when using papermill is **parameterizing** a notebook, **executing** it, as well as **storing** the results. In addition to operating on a single notebook, papermill also works on a collection of notebooks.

Parameterize

> **See also**
>
> *Workflow reference*

Generally, the first workflow step when using papermill is to parameterize the notebook.

To do this, tag notebook cells with `parameters`. These `parameters` are later used when the notebook is executed or run.

Designate parameters for a cell

To parameterize a notebook, designate a cell with the tag `parameters`.

Notebook

If using the Jupyter Notebook interface

1. Activate the tagging toolbar by navigating to `View`, `Cell Toolbar`, and then `Tags`
2. Enter `parameters` into a textbox at the top right of a cell
3. Click `Add tag`

JupyterLab 3.0+

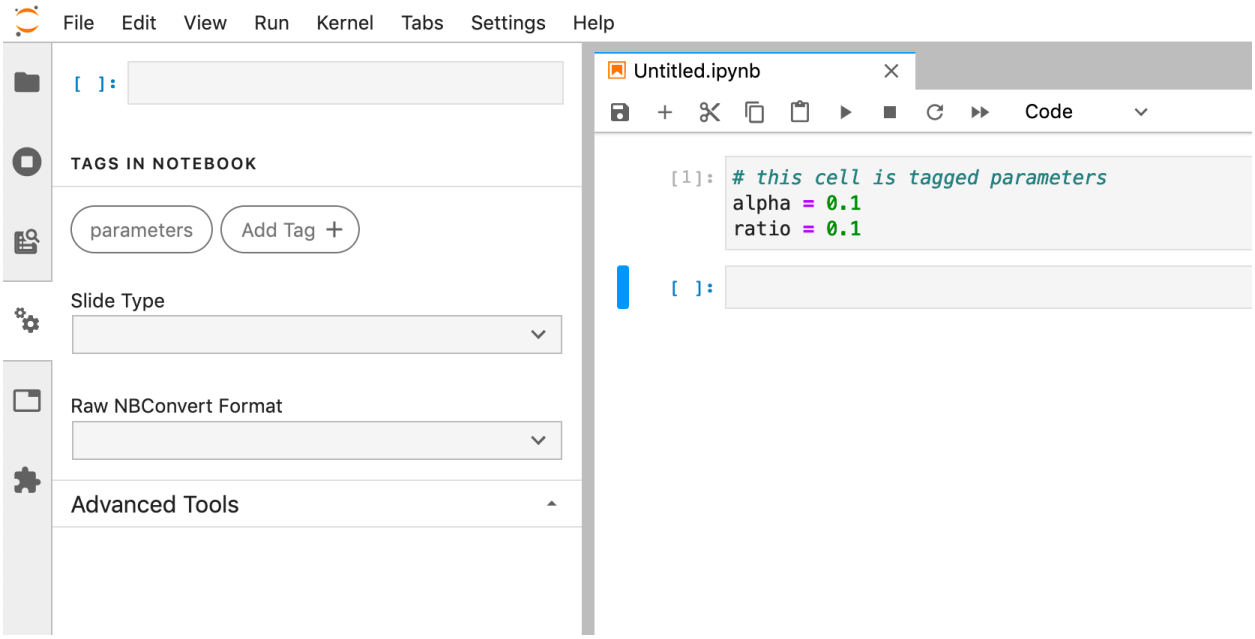
If using JupyterLab v3 or above:

1. Select the cell to parameterize
2. Click the property inspector in the right sidebar (double gear icon)
3. Type “parameters” in the “Add Tag” box and hit “Enter”.

JupyterLab 2.0 - 2.2.x

If using the JupyterLab interface

1. Select the cell to parameterize
2. Click the property inspector on the left side (double gear icon)
3. Type “parameters” in the “Add Tag” box and hit “Enter”.



JupyterLab < 2.0

If using JupyterLab < 2.0, consider using the jupyterlab-celltags extension.

1. Select the cell to parameterize
2. Click the cell inspector (wrench icon)
3. Type "parameters" in the "Add Tag" box and hit "Enter".

If the extension is not installed, you can manually add the tag by editing the Cell Metadata field in the cell inspector by adding "tags":["parameters"].

Learn more about the jupyter notebook format and metadata fields here.

How parameters work

The `parameters` cell is assumed to specify default values which may be overridden by values specified at execution time.

- papermill inserts a new cell tagged `injected-parameters` immediately after the `parameters` cell
- `injected-parameters` contains only the overridden parameters
- subsequent cells are treated as normal cells, even if also tagged `parameters`
- if no cell is tagged `parameters`, the `injected-parameters` cell is inserted at the top of the notebook

One caveat is that a `parameters` cell may not behave intuitively with inter-dependent parameters. Consider a notebook `note.ipynb` with two cells:

```
#parameters
a = 1
twice = a * 2
```

```
print("a =", a, "and twice =", twice)
```

when executed with `papermill note.ipynb -p a 9`, the output will be `a = 9 and twice = 2` (not `twice = 18`).

Inspect

The two ways to inspect the notebook to discover its parameters are: (1) through the Python API and (2) through the command line interface.

Execute via the Python API

The *inspect_notebook* function can be called to inspect a notebook:

```
inspect_notebook(<notebook path>)
```

```
import papermill as pm

pm.inspect_notebook('path/to/input.ipynb')
```

Note

If your path is parameterized, you can pass those parameters in a dictionary as second parameter:

`inspect_notebook('path/to/input_{month}.ipynb', parameters={month='Feb'})`

Inspect via CLI

To inspect a notebook using the CLI, enter the `papermill --help-notebook` command in the terminal with the notebook and optionally path parameters.

See also

CLI reference

Inspect a notebook

Here's an example of a local notebook being inspected and an output example:

```
papermill --help-notebook ./papermill/tests/notebooks/complex_parameters.ipynb

Usage: papermill [OPTIONS] NOTEBOOK_PATH [OUTPUT_PATH]

Parameters inferred for notebook './papermill/tests/notebooks/complex_parameters.ipynb':
msg: Unknown type (default None)
a: float (default 2.25)          Variable a
b: List[str] (default ['Hello','World'])
                                 Nice list
c: NoneType (default None)
```

Execute

The two ways to execute the notebook with parameters are: (1) through the Python API and (2) through the command line interface.

Execute via the Python API

The *execute_notebook* function can be called to execute an input notebook when passed a dictionary of parameters:

```
execute_notebook(<input notebook>, <output notebook>, <dictionary of parameters>)
```

```
import papermill as pm

pm.execute_notebook(
   'path/to/input.ipynb',
   'path/to/output.ipynb',
   parameters=dict(alpha=0.6, ratio=0.1)
)
```

Execute via CLI

To execute a notebook using the CLI, enter the `papermill` command in the terminal with the input notebook, location for output notebook, and options.

See also

CLI reference

Execute a notebook with parameters

Here's an example of a local notebook being executed and output to an Amazon S3 account:

```
$ papermill local/input.ipynb s3://bkt/output.ipynb -p alpha 0.6 -p l1_ratio 0.1
```

In the above example, two parameters are set: `alpha` and `l1_ratio` using `-p` (`--parameters` also works). Parameter values that look like booleans or numbers will be interpreted as such.

Here are the different ways users may set parameters:

Using raw strings as parameters

Using `-r` or `--parameters_raw`, users can set parameters one by one. However, unlike `-p`, the parameter will remain a string, even if it may be interpreted as a number or boolean.

```
$ papermill local/input.ipynb s3://bkt/output.ipynb -r version 1.0
```

Using a parameters file

Using `-f` or `--parameters_file`, users can provide a YAML file from which parameter values should be read.

```
$ papermill local/input.ipynb s3://bkt/output.ipynb -f parameters.yaml
```

Using a YAML string for parameters

Using `-y` or `--parameters_yaml`, users can directly provide a YAML string containing parameter values.

```
$ papermill local/input.ipynb s3://bkt/output.ipynb -y "
x:
    - 0.0
    - 1.0
    - 2.0
    - 3.0
linear_function:
    slope: 3.0
    intercept: 1.0"
```

Using `-b` or `--parameters_base64`, users can provide a YAML string, base64-encoded, containing parameter values.

```
$ papermill local/input.ipynb s3://bkt/output.ipynb -b␣
↪YWxwaGE6IDAuNgpsMV9yYXRpbzogMC4xCg==
```

Note about using YAML

When using YAML to pass arguments, through `-y`, `-b` or `-f`, parameter values can be arrays or dictionaries:

```
$ papermill local/input.ipynb s3://bkt/output.ipynb -y "
x:
    - 0.0
    - 1.0
    - 2.0
    - 3.0
linear_function:
    slope: 3.0
    intercept: 1.0"
```

Note about using with multiple account credentials

If you use multiple AWS accounts and are accessing S3 files, you can configure your AWS credentials, to specify which account to use by setting the *AWS_PROFILE* environment variable at the command-line. For example:

```
$ AWS_PROFILE=dev_account papermill local/input.ipynb s3://bkt/output.ipynb -p alpha 0.6␣
↪-p l1_ratio 0.1
```

A similar pattern may be needed for other types of remote storage accounts.

Tracking the Execution with Cell Descriptions

If you want to keep track of the execution in a closer way, you can add cell descriptions to the notebook being executed that will be injected to the TQDM progress bar.

Here is an example of the TQDM output during the execution of a 4-cells notebook without cell descriptions:

```
10:10 # papermill input_notebook.ipynb output_notebook.ipynb
Input Notebook:  input_notebook.ipynb
Output Notebook: output_notebook.ipynb
Executing:   0%|                                                                    | 0/
```

(continues on next page)

(continued from previous page)

```
↪4 [00:00<?, ?cell/s]Executing
Executing:  25%|                                                    | 1/4 [00:00<?
↪, ?cell/s]Executing
[...]
```

You can inject cell descriptions by adding a comment with the following formatting at the very beginning of each cell you are interested in:

```
#papermill_description=TQDM_DESCRIPTION
```

In this way, when the execution will reach the cell tagged with the description, the TQDM progress bar will show the string TQDM_DESCRIPTION. Please be careful: the comment must not contain any space, otherwise it will be ignored.

Taking back our previous example, if we add the description 'FirstCell' to cell 0 and the description 'SecondCell' to the cell 1 we will get the TQDM to display the following information:

```
10:12 # papermill input_notebook.ipynb output_notebook.ipynb
Input Notebook:  input_notebook.ipynb
Output Notebook: output_notebook.ipynb
Executing FirstCell:   0%|                                                  | 0/
↪4 [00:00<?, ?cell/s]Executing
Executing SecondCell:  25%|                                               | 1/4 [00:00
↪<?, ?cell/s]Executing
[...]
```

Store

See also

Reference - Storage

Papermill can store notebooks in a number of locations including AWS S3, Azure data blobs, and Azure data lakes.

The modular architecture of papermill allows new data stores to be added over time.

2.3 Command Line Interface

papermill may be executed from the terminal. The following are the command options:

```
Usage: papermill [OPTIONS] NOTEBOOK_PATH [OUTPUT_PATH]

  This utility executes a single notebook in a subprocess.

  Papermill takes a source notebook, applies parameters to the source
  notebook, executes the notebook with the specified kernel, and saves the
  output in the destination notebook.

  The NOTEBOOK_PATH and OUTPUT_PATH can now be replaced by `-` representing
  stdout and stderr, or by the presence of pipe inputs / outputs. Meaning
  that
```

(continues on next page)

(continued from previous page)

```
  `<generate input>... | papermill | ...<process output>`

  with `papermill - -` being implied by the pipes will read a notebook from
  stdin and write it out to stdout.

Options:
  --help-notebook                 Display parameters information for the given
                                  notebook path.

  -p, --parameters TEXT...        Parameters to pass to the parameters cell.
  -r, --parameters_raw TEXT...    Parameters to be read as raw string.
  -f, --parameters_file TEXT      Path to YAML file containing parameters.
  -y, --parameters_yaml TEXT      YAML string to be used as parameters.
  -b, --parameters_base64 TEXT    Base64 encoded YAML string as parameters.
  --inject-input-path             Insert the path of the input notebook as
                                  PAPERMILL_INPUT_PATH as a notebook
                                  parameter.

  --inject-output-path            Insert the path of the output notebook as
                                  PAPERMILL_OUTPUT_PATH as a notebook
                                  parameter.

  --inject-paths                  Insert the paths of input/output notebooks
                                  as
                                  PAPERMILL_INPUT_PATH/PAPERMILL_OUTPUT_PATH
                                  as notebook parameters.

  --engine TEXT                   The execution engine name to use in
                                  evaluating the notebook.

  --request-save-on-cell-execute / --no-request-save-on-cell-execute
                                  Request save notebook after each cell
                                  execution

  --autosave-cell-every INTEGER   How often in seconds to autosave the
                                  notebook during long cell executions (0 to
                                  disable)

  --prepare-only / --prepare-execute
                                  Flag for outputting the notebook without
                                  execution, but with parameters applied.

  -k, --kernel TEXT               Name of kernel to run.
  --cwd TEXT                      Working directory to run notebook in.
  --progress-bar / --no-progress-bar
                                  Flag for turning on the progress bar.
  --log-output / --no-log-output  Flag for writing notebook output to the
                                  configured logger.

  --stdout-file FILENAME          File to write notebook stdout output to.
  --stderr-file FILENAME          File to write notebook stderr output to.
```

(continues on next page)

(continued from previous page)

```
--log-level [NOTSET|DEBUG|INFO|WARNING|ERROR|CRITICAL]
                                Set log level
--start-timeout, --start_timeout INTEGER
                                Time in seconds to wait for kernel to start.
--execution-timeout INTEGER     Time in seconds to wait for each cell before
                                failing execution (default: forever)

--report-mode / --no-report-mode
                                Flag for hiding input.
--version                       Flag for displaying the version.
-h, --help                      Show this message and exit.
```

2.4 Extending papermill

Papermill provides some interfaces with external services out of the box. However, you may find that you would like papermill to do more than it currently does. You could contribute to the papermill project yourself (see *Extending papermill by contributing to it*). However, an easier method might be to extend papermill using entry points.

In general, when you run a notebook with papermill, the following happens:

1. The notebook file is read in
2. The file content is converted to a notebook python object
3. The notebook is executed
4. The notebook is written to a file

Through entry points, you can write your own tools to handle steps 1, 3, and 4. If you find that there's more you want to contribute to papermill, consider developing papermill itself.

2.4.1 Extending papermill through entry points

What are entry points?

The python packaging documentation describes entry points as:

> Entry points are a mechanism for an installed distribution to advertise components it provides to be discovered and used by other code. For example:
>
> Distributions can specify console_scripts entry points, each referring to a function. When pip (or another console_scripts aware installer) installs the distribution, it will create a command-line wrapper for each entry point.
>
> Applications can use entry points to load plugins; e.g. Pygments (a syntax highlighting tool) can use additional lexers and styles from separately installed packages. For more about this, see Creating and discovering plugins.

When running, papermill looks for entry points that implement input / output (I/O) handlers, and execution handlers.

Developing new I/O handlers

Virtually the first thing that happens when papermill is used is that the input notebook is read in. This is managed by I/O handlers, which allow papermill to access not just the local filesystem, but also remote services such as Amazon S3. The same goes for writing the executed notebook to a file system: I/O handlers allow papermill to write files to S3 or otherwise.

Creating a new handler

Writing your own I/O handler requires writing a class that has four methods. All I/O handlers should implement the following class methods:

- `CustomIO.read(file_path)`, returning the file content
- `CustomIO.write(file_content, file_path)`, returning nothing
- `CustomIO.pretty_path(path)`, returning a prettified path
- `CustomIO.listdir(path)`, returning a list of paths.

> **Note**
>
> If you don't want to support things such as `read` because your I/O handler is only intended for writing (such as a publish-only platform), then you should implement the method but raise an exception when it is used.

Ensuring your handler is found by papermill

Once you have developed a new handler, you need to declare papermill entry points in your `pyproject.toml` file.

This is done by including the `[project.entry-points."papermill.io"]` section in your pyproject.toml file:

```
[project.entry-points."papermill.io"]
"sftp://" = "papermill_sftp:SFTPHandler"
```

This indicates to papermill that when a file path begins with `sftp://`, it should use the class `papermill_sftp.SFTPHandler` to handle reading or writing to that path. Anything before the equal sign is the path prefix, and everything after it is the class to be used, including where it is imported from.

Traditionally, entry points for papermill I/O handlers look like URL prefixes. For example, the Amazon Web Services S3 handler is registered under `s3://`, and so is used whenever a path begins with `s3://`.

Example: sftp I/O handler

As an example, let's go through how we would create an I/O handler that reads from an sftp server and writes back to it, so we could do the following:

```
papermill sftp://my_ftp_server.co.uk/input.ipynb sftp://my_ftp_server.co.uk/output.ipynb
```

Our project structure will look like this:

```
papermill_sftp
    |- pyproject.toml
    |- src
        |- papermill_sftp
            |- __init__.py
```

We can define the I/O handler in `src/papermill_sftp/__init__.py`. To do so, we have to create a class that does the relevant actions.

For reading, we will download the file to a temporary path and read it in from there. For writing, we will write to a temporary path and upload it from there. Prettifying the path doesn't need to change the path, and we are not going to implement a listdir option for now.

```python
import os
import pysftp

sftp_username = os.getenv('SFTP_USERNAME')
sftp_password = os.getenv('SFTP_PASSWORD')

class SFTPHandler:

    @classmethod
    def read(cls, path):
        """
        Read a notebook from an SFTP server.
        """
        parsed_url = urllib.parse.urlparse(path)
        with tempfile.TemporaryDirectory() as tmpdir:
            tmp_file = pathlib.Path(tmpdir) / pathlib.Path(parsed_url.path).name
            with pysftp.Connection(
                parsed_url.hostname,
                username=sftp_username,
                password=sftp_password,
                port=(parsed_url.port or 22),
                cnopts=cnopts,
            ) as sftp:
                sftp.get(parsed_url.path, str(tmp_file))
            return tmp_file.read_text()

    @classmethod
    def write(cls, file_content, path):
        """
        Write a notebook to an SFTP server.
        """
        parsed_url = urllib.parse.urlparse(path)
        with tempfile.TemporaryDirectory() as tmpdir:
            tmp_file = pathlib.Path(tmpdir) / "output.ipynb"
            tmp_file.write_text(file_content)
            with pysftp.Connection(
                parsed_url.hostname,
                username=sftp_username,
                password=sftp_password,
                port=(parsed_url.port or 22),
                cnopts=cnopts,
            ) as sftp:
                sftp.put(str(tmp_file), parsed_url.path)

    @classmethod
    def pretty_path(cls, path):
        return path

    @classmethod
    def listdir(cls, path):
        raise NotImplementedError
```

The `pyproject.toml` file contains the following code:

```
[build-system]
requires = ["setuptools>=61.0", "wheel"]
build-backend = "setuptools.build_meta"

[project]
name = "papermill_sftp"
version = "0.1"
description = "An SFTP I/O handler for papermill."
authors = [
    {name = "My Name", email = "my.email@gmail.com"}
]
dependencies = ["pysftp"]

[project.urls]
Repository = "https://github.com/my_username/papermill_sftp.git"

[project.entry-points."papermill.io"]
"sftp://" = "papermill_sftp:SFTPHandler"

[tool.setuptools]
packages = ["papermill_sftp"]
package-dir = {"" = "src"}
```

When executing, papermill will check if the input or output path begin with `sftp://`, and if so, use the SFTPHandler from the papermill_sftp project.

Developing a new engine

A papermill engine is a python object that can run, or execute, a notebook. The default implementation in papermill for example takes in a notebook object, and runs it locally on your machine.

By writing a custom engine, you could allow execution to be handled remotely, or you could apply post-processing to the executed notebook. In the next section, you will see a demonstration.

Creating a new engine

Papermill engines need to inherit from the `papermill.engines.Engine` class.

In order to be used, the new class needs to implement the class method `execute_managed_notebook`. The call signature should match that of the parent class:

```
class CustomEngine(papermill.engines.Engine):

    @classmethod
    def execute_managed_notebook(cls, nb_man, kernel_name, **kwargs):
        pass
```

`nb_man` is a `nbformat.NotebookNode` object, and `kernel_name` is a string. Your custom class then needs to implement the execution of the notebook. For example, you could insert code that executes the notebook remotely on a server, or executes the notebook many times to simulate different conditions.

As an example, the following project implements a custom engine that adds the time it took to execute each cell as additional output after every code cell.

The project structure is:

```
papermill_timing
    |- pyproject.toml
    |- src
        |- papermill_timing
            |- __init__.py
```

The file `src/papermill_timing/__init__.py` will implement the engine. Since papermill already stores information about execution timing in the metadata, we can leverage the default engine. We will also need to use the `nbformat` library to create a notebook node object.

```
from datetime import datetime
from papermill.engines import NBClientEngine
from nbformat.v4 import new_output

class CustomEngine(NBClientEngine):

    @classmethod
    def execute_managed_notebook(cls, nb_man, kernel_name, **kwargs):

        # call the papermill execution engine:
        super().execute_managed_notebook(nb_man, kernel_name, **kwargs)

        for cell in nb_man.nb.cells:

            if cell.cell_type == "code" and cell.execution_count is not None:
                start = datetime.fromisoformat(cell.metadata.papermill.start_time)
                end = datetime.fromisoformat(cell.metadata.papermill.end_time)
                output_message = f"Execution took {(end - start).total_seconds():.3f}␣
↪seconds"
                output_node = new_output("display_data", data={"text/plain": [output_
↪message]})
                cell.outputs = [output_node] + cell.outputs
```

Once this is in place, we need to add our engine as an entry point to our `pyproject.toml` file - for this, see the following section.

Ensuring your engine is found by papermill

Custom engines can be specified as entry points, under the `papermill.engine` prefix. The entry point needs to reference the class that we have just implemented. For example, if you write an engine called TimingEngine in a package called papermill_timing, then in the `pyproject.toml` file, you should specify:

```
[build-system]
requires = ["setuptools>=61.0", "wheel"]
build-backend = "setuptools.build_meta"

[project]
name = "papermill_timing"
version = "0.1"
description = "A papermill engine that logs additional timing information about code."
authors = [
    {name = "My Name", email = "my.email@gmail.com"}
]
```

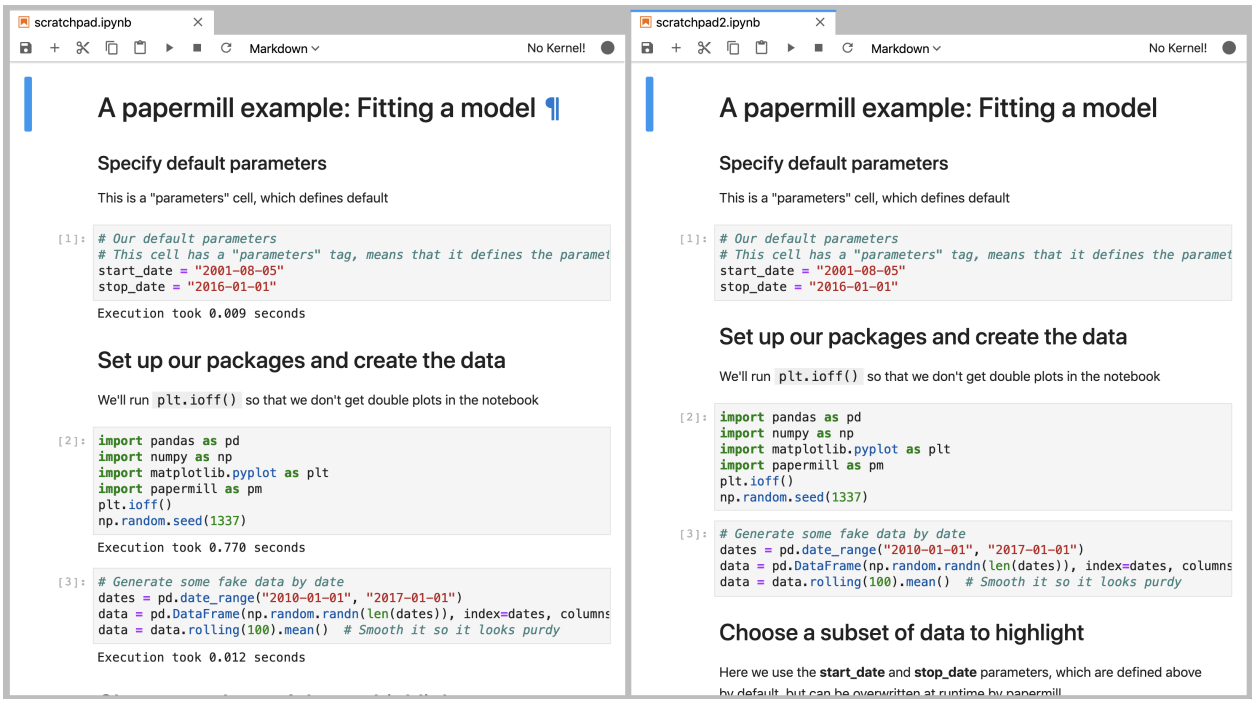
(continues on next page)

(continued from previous page)

```
dependencies = ["papermill", "nbformat"]

[project.urls]
Repository = "https://github.com/my_username/papermill_timing.git"

[project.entry-points."papermill.engine"]
timer_engine = "papermill_timing:CustomEngine"

[tool.setuptools]
packages = ["papermill_timing"]
package-dir = {"" = "src"}
```

This allows users to specify the engine from `papermill_timing` by passing the command line argument `--engine timer_engine`.

In the image below, the notebook on the left was executed with the new custom engine, while the one on the left was executed with the standard papermill engine. As you can see, this adds our "injected" output to each code cell



2.4.2 Extending papermill by contributing to it

If you find that you'd like to not only add I/O and execution handlers, but think a fundamental aspect of the project could use some improvement, then you may want to contribute to it.

Development of papermill happens on github, and a detailed guide to contributing to it can be found there. There is also a code of conduct there. Please read both documents before beginning!

2.5 Troubleshooting

2.5.1 NoSuchKernel Errors (using Conda)

`NoSuchKernel` Errors can appear when running papermill on jupyter notebooks whose kernel has been specified via conda (nb_conda). nb_conda is used to easily set conda environment per notebook from within jupyterlab.

To illustrate, the following example demonstrates the creation of a new environment with all the dependencies necessary for an analysis.

```
conda create -n analysis_1 python=2 ipykernel
```

Once nb_conda is used within the jupyter server to set the kernel for a notebook to *analysis_1*, the notebook gets metadata similar to the following:

```
{
 "kernelspec": {
  "display_name": "Python [conda env:analysis_1]",
  "language": "python",
  "name": "conda-env-analysis_1-py"
 }
}
```

Papermill cannot use this metadata to determine that it should use *analysis_1* to execute this notebook. Running papermill (from *analysis_1* or another environment) will raise the following error:

```
jupyter_client.kernelspec.NoSuchKernel: No such kernel named conda-env-analysis_1-py
```

This can be fixed by:

- Installing jupyter (or at least ipykernel) in *analysis_1*

```
conda install -n analysis_1 jupyter
```

- Expose the *analysis_1* environment as a jupyter kernel (this is no longer automatic).

```
conda activate analysis_1
jupyter kernelspec install --user --name analysis_1
```

- Run papermill (from any environment) specifying the correct kernel using the `-k` option

```
papermill my_notebook.ipynb output_notebook.ipynb -k analysis_1
```

2.6 Change Log

2.6.1 2.7.0

Highlights

- Drop Python 3.8 and 3.9, add Python 3.13 support
- Drop `ansicolors` dependency
- Modernize packaging to use `pyproject.toml`

Changes

- Add Python 3.13 support, drop Python 3.8/3.9 #828
- Drop the dependency on `ansicolors` #822
- Replace deprecated `datetime.utcnow()` #823
- Modernize packaging to use `pyproject.toml` #837
- Changed parameter inspection to raise the same error messages as other pathways for missing kernel name and language

Bug Fixes

- Fix failing tests in CI, pin `azure-datalake-store` #826
- Fix skipped HDFS tests for Python 3.12 #839
- Unskip tests that were previously failing #846

Dev / CI

- Add `pyproject-fmt` and `validate-pyproject` pre-commit hooks #858
- Add Dependabot for GitHub Actions #857
- Update pre-commit config #843
- Update CONTRIBUTING.md #842 #847
- Update docs/RTD configuration #805 #836
- Bump CI actions to latest versions #852 #853 #854 #855

2.6.2 2.6.0

- bring back strip_color and remove ANSI color codes from exception traceback #791
- cleaned up documentation #790
- prevent error override, fix traceback type #788
- Upgrade tests to moto v5 #779
- raise PapermillExecutionError when CellExecutionError is raised without cell error output #786
- make progress_bar param accept a dict #778
- Fix nbformat to 5.2.0 to cell None type #770
- Use f-strings where possible #762
- Unmark wheel as universal #764

CHAPTER

THREE

API REFERENCE

If you are looking for information about a specific function, class, or method, this documentation section will help you.

3.1 Reference

This part of the documentation lists the full API reference of all public classes and functions.

3.1.1 CLI

papermill.cli

Main *papermill* interface.

papermill.cli.**print_papermill_version**(*ctx*, *param*, *value*)

Command Line options

```
Usage: papermill [OPTIONS] NOTEBOOK_PATH OUTPUT_PATH

  This utility executes a single notebook in a subprocess.

  Papermill takes a source notebook, applies parameters to the source
  notebook, executes the notebook with the specified kernel, and saves the
  output in the destination notebook.

  The NOTEBOOK_PATH and OUTPUT_PATH can now be replaced by `-` representing
  stdout and stderr, or by the presence of pipe inputs / outputs. Meaning
  that

  `<generate input>... | papermill | ...<process output>`

  with `papermill - -` being implied by the pipes will read a notebook from
  stdin and write it out to stdout.

Options:
  -p, --parameters TEXT...        Parameters to pass to the parameters cell.
  -r, --parameters_raw TEXT...    Parameters to be read as raw string.
  -f, --parameters_file TEXT      Path to YAML file containing parameters.
  -y, --parameters_yaml TEXT      YAML string to be used as parameters.
  -b, --parameters_base64 TEXT    Base64 encoded YAML string as parameters.
  --inject-input-path             Insert the path of the input notebook as
```

(continues on next page)

(continued from previous page)

```
                                PAPERMILL_INPUT_PATH as a notebook
                                parameter.
  --inject-output-path          Insert the path of the output notebook as
                                PAPERMILL_OUTPUT_PATH as a notebook
                                parameter.
  --inject-paths                Insert the paths of input/output notebooks
                                as
                                PAPERMILL_INPUT_PATH/PAPERMILL_OUTPUT_PATH
                                as notebook parameters.
  --engine TEXT                 The execution engine name to use in
                                evaluating the notebook.
  --request-save-on-cell-execute / --no-request-save-on-cell-execute
                                Request save notebook after each cell
                                execution
  --prepare-only / --prepare-execute
                                Flag for outputting the notebook without
                                execution, but with parameters applied.
  -k, --kernel TEXT             Name of kernel to run.
  --cwd TEXT                    Working directory to run notebook in.
  --progress-bar / --no-progress-bar
                                Flag for turning on the progress bar.
  --log-output / --no-log-output  Flag for writing notebook output to the
                                configured logger.
  --stdout-file FILENAME        File to write notebook stdout output to.
  --stderr-file FILENAME        File to write notebook stderr output to.
  --log-level [NOTSET|DEBUG|INFO|WARNING|ERROR|CRITICAL]
                                Set log level
  --start_timeout INTEGER       Time in seconds to wait for kernel to start.
  --execution_timeout INTEGER   Time in seconds to wait for each cell before
                                failing execution (default: forever)
  --report-mode / --no-report-mode
                                Flag for hiding input.
  --version                     Flag for displaying the version.
  -h, --help                    Show this message and exit.
```

3.1.2 Workflow

papermill.engines

Engines to perform different roles

class papermill.engines.**Engine**

Bases: object

Base class for engines.

Other specific engine classes should inherit and implement the *execute_managed_notebook* method.

Defines *execute_notebook* method which is used to correctly setup the *NotebookExecutionManager* object for engines to interact against.

classmethod execute_managed_notebook(*nb_man*, *kernel_name*, ***kwargs*)

An abstract method where implementation will be defined in a subclass.

classmethod execute_notebook(*nb*, *kernel_name*, *output_path=None*, *progress_bar=True*, *log_output=False*, *autosave_cell_every=30*, ***kwargs*)

A wrapper to handle notebook execution tasks.

Wraps the notebook object in a *NotebookExecutionManager* in order to track execution state in a uniform manner. This is meant to help simplify engine implementations. This allows a developer to just focus on iterating and executing the cell contents.

classmethod nb_kernel_name(*nb*, *name=None*)

Use default implementation to fetch kernel name from the notebook object

classmethod nb_language(*nb*, *language=None*)

Use default implementation to fetch programming language from the notebook object

class papermill.engines.**NBClientEngine**

Bases: *Engine*

A notebook engine representing an nbclient process.

This can execute a notebook document and update the *nb_man.nb* object with the results.

classmethod execute_managed_notebook(*nb_man*, *kernel_name*, *log_output=False*, *stdout_file=None*, *stderr_file=None*, *start_timeout=60*, *execution_timeout=None*, ***kwargs*)

Performs the actual execution of the parameterized notebook locally.

Parameters

- **nb_man** (NotebookExecutionManager) – Wrapper for execution state of a notebook.
- **kernel_name** (*str*) – Name of kernel to execute the notebook against.
- **log_output** (*bool*) – Flag for whether or not to write notebook output to the configured logger.
- **start_timeout** (*int*) – Duration to wait for kernel start-up.
- **execution_timeout** (*int*) – Duration to wait before failing execution (default: never).

class papermill.engines.**NotebookExecutionManager**(*nb*, *output_path=None*, *log_output=False*, *progress_bar=True*, *autosave_cell_every=30*)

Bases: object

Wrapper for execution state of a notebook.

This class is a wrapper for notebook objects to house execution state related to the notebook being run through an engine.

In particular the NotebookExecutionManager provides common update callbacks for use within engines to facilitate metadata and persistence actions in a shared manner.

COMPLETED = 'completed'

FAILED = 'failed'

PENDING = 'pending'

RUNNING = 'running'

autosave_cell()

Saves the notebook if it's been more than self.autosave_cell_every seconds since it was last saved.

cell_complete(*cell*, *cell_index=None*, ***kwargs*)

Finalize metadata for a cell and save notebook.

Optionally called by engines during execution to finalize the metadata for a cell and save the notebook to the output path.

cell_exception(*cell*, *cell_index=None*, ***kwargs*)

Set metadata when an exception is raised.

Called by engines when an exception is raised within a notebook to set the metadata on the notebook indicating the location of the failure.

cell_start(*cell*, *cell_index=None*, ***kwargs*)

Set and save a cell's start state.

Optionally called by engines during execution to initialize the metadata for a cell and save the notebook to the output path.

cleanup_pbar()

Clean up a progress bar

complete_pbar()

Refresh progress bar

get_cell_description(*cell*, *escape_str='papermill_description='*)

Fetches cell description if present

notebook_complete(***kwargs*)

Finalize the metadata for a notebook and save the notebook to the output path.

Called by Engine when execution concludes, regardless of exceptions.

notebook_start(***kwargs*)

Initialize a notebook, clearing its metadata, and save it.

When starting a notebook, this initializes and clears the metadata for the notebook and its cells, and saves the notebook to the given output path.

Called by Engine when execution begins.

now()

Helper to return current UTC time

save(***kwargs*)

Saves the wrapped notebook state.

If an output path is known, this triggers a save of the wrapped notebook state to the provided path.

Can be used outside of cell state changes if execution is taking a long time to conclude but the notebook object should be synced.

For example, you may want to save the notebook every 10 minutes when running a 5 hour cell execution to capture output messages in the notebook.

set_timer()

Initializes the execution timer for the notebook.

This is called automatically when a NotebookExecutionManager is constructed.

class papermill.engines.**PapermillEngines**

Bases: object

The holder which houses any engine registered with the system.

This object is used in a singleton manner to save and load particular named Engine objects so they may be referenced externally.

execute_notebook_with_engine(*engine_name*, *nb*, *kernel_name*, ***kwargs*)

Fetch a named engine and execute the nb object against it.

get_engine(*name=None*)

Retrieves an engine by name.

nb_kernel_name(*engine_name*, *nb*, *name=None*)

Fetch kernel name from the document by dropping-down into the provided engine.

nb_language(*engine_name*, *nb*, *language=None*)

Fetch language from the document by dropping-down into the provided engine.

register(*name*, *engine*)

Register a named engine

register_entry_points()

Register entrypoints for an engine

Load handlers provided by other packages

papermill.engines.**catch_nb_assignment**(*func*)

Wrapper to catch *nb* keyword arguments

This helps catch *nb* keyword arguments and assign onto self when passed to the wrapped function.

Used for callback methods when the caller may optionally have a new copy of the originally wrapped *nb* object.

papermill.execute

papermill.execute.**execute_notebook**(*input_path*, *output_path*, *parameters=None*, *engine_name=None*, *request_save_on_cell_execute=True*, *prepare_only=False*, *kernel_name=None*, *language=None*, *progress_bar=True*, *log_output=False*, *stdout_file=None*, *stderr_file=None*, *start_timeout=60*, *report_mode=False*, *cwd=None*, ***engine_kwargs*)

Executes a single notebook locally.

Parameters

- **input_path** (*str or Path or nbformat.NotebookNode*) – Path to input notebook or NotebookNode object of notebook
- **output_path** (*str or Path or None*) – Path to save executed notebook. If None, no file will be saved
- **parameters** (*dict, optional*) – Arbitrary keyword arguments to pass to the notebook parameters
- **engine_name** (*str, optional*) – Name of execution engine to use
- **request_save_on_cell_execute** (*bool, optional*) – Request save notebook after each cell execution
- **autosave_cell_every** (*int, optional*) – How often in seconds to save in the middle of long cell executions

- **prepare_only** (*bool, optional*) – Flag to determine if execution should occur or not
- **kernel_name** (*str, optional*) – Name of kernel to execute the notebook against
- **language** (*str, optional*) – Programming language of the notebook
- **progress_bar** (*bool, optional*) – Flag for whether or not to show the progress bar.
- **log_output** (*bool, optional*) – Flag for whether or not to write notebook output to the configured logger
- **start_timeout** (*int, optional*) – Duration in seconds to wait for kernel start-up
- **report_mode** (*bool, optional*) – Flag for whether or not to hide input.
- **cwd** (*str or Path, optional*) – Working directory to use when executing the notebook
- ****kwargs** – Arbitrary keyword arguments to pass to the notebook engine

Returns
nb – Executed notebook object

Return type
NotebookNode

papermill.execute.**prepare_notebook_metadata**(*nb*, *input_path*, *output_path*, *report_mode=False*)

Prepare metadata associated with a notebook and its cells

Parameters

- **nb** (*NotebookNode*) – Executable notebook object
- **input_path** (*str*) – Path to input notebook
- **output_path** (*str*) – Path to write executed notebook
- **report_mode** (*bool, optional*) – Flag to set report mode

papermill.execute.**raise_for_execution_errors**(*nb*, *output_path*)

Assigned parameters into the appropriate place in the input notebook

Parameters

- **nb** (*NotebookNode*) – Executable notebook object
- **output_path** (*str*) – Path to write executed notebook

papermill.execute.**remove_error_markers**(*nb*)

papermill.clientwrap

class papermill.clientwrap.**PapermillNotebookClient**(***kwargs: Any*)

Bases: NotebookClient

Module containing a that executes the code cells and updates outputs

execute(***kwargs*)

Wraps the parent class process call slightly

log_output

A boolean (True, False) trait.

`log_output_message`(*output*)

Process a given output. May log it in the configured logger and/or write it into the configured stdout/stderr files.

Parameters
output – nbformat.notebooknode.NotebookNode

Returns

`papermill_execute_cells`()

This function replaces cell execution with it's own wrapper.

We are doing this for the following reasons:

1. Notebooks will stop executing when they encounter a failure but not raise a *CellException*. This allows us to save the notebook with the traceback even though a *CellExecutionError* was encountered.
2. We want to write the notebook as cells are executed. We inject our logic for that here.
3. We want to include timing and execution status information with the metadata of each cell.

`process_message`(**arg*, ***kwargs*)

Processes a kernel message, updates cell state, and returns the resulting output object that was appended to cell.outputs.

The input argument *cell* is modified in-place.

Parameters

- **msg** (*dict*) – The kernel message being processed.
- **cell** (*nbformat.NotebookNode*) – The cell which is currently being processed.
- **cell_index** (*int*) – The position of the cell within the notebook object.

Returns
output – The execution output payload (or None for no output).

Return type
NotebookNode

Raises
CellExecutionComplete – Once a message arrives which indicates computation completeness.

`stderr_file`

A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

`stdout_file`

A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

3.1.3 Language Translators

Translators

Translator

class papermill.translators.**Translator**

classmethod assign(*name*, *str_val*)

classmethod codify(*parameters*, *comment='Parameters'*)

classmethod comment(*cmt_str*)

classmethod inspect(*parameters_cell*)

Inspect the parameters cell to get a Parameter list

It must return an empty list if no parameters are found and it should ignore inspection errors.

> **Note**
>
> `inferred_type_name` should be "None" if unknown (set it to "NoneType" for null value)

Parameters
parameters_cell (*NotebookNode*) – Cell tagged _parameters_

Returns
A list of all parameters

Return type
List[Parameter]

classmethod translate(*val*)

Translate each of the standard json/yaml types to appropriate objects.

classmethod translate_bool(*val*)

Default behavior for translation

classmethod translate_dict(*val*)

classmethod translate_escaped_str(*str_val*)

Reusable by most interpreters

classmethod translate_float(*val*)

Default behavior for translation

classmethod translate_int(*val*)

Default behavior for translation

classmethod translate_list(*val*)

classmethod translate_none(*val*)

Default behavior for translation

classmethod translate_raw_str(*val*)

Reusable by most interpreters

classmethod translate_str(*val*)

Default behavior for translation

PapermillTranslators

class papermill.translators.**PapermillTranslators**

The holder which houses any translator registered with the system. This object is used in a singleton manner to save and load particular named Translator objects for reference externally.

find_translator(*kernel_name*, *language*)

register(*language*, *translator*)

Python

class papermill.translators.**PythonTranslator**

PARAMETER_PATTERN = re.compile('^(?P<target>\\w[\\w_]*)\\s*(:\\s*[\\"\']?(?P<annotation>\\w[\\w_\\[\\],\\s]*)[\\"\']?\\s*)?=\\s*(?P<value>.*?)(\\s*#\\s*(type:\\s*(?P<type_comment>[^\\s]*)\\s*)?(?P<help>.*))?$')

classmethod codify(*parameters*, *comment='Parameters'*)

classmethod comment(*cmt_str*)

classmethod inspect(*parameters_cell*)

Inspect the parameters cell to get a Parameter list

It must return an empty list if no parameters are found and it should ignore inspection errors.

Parameters
parameters_cell (*NotebookNode*) – Cell tagged _parameters_

Returns
A list of all parameters

Return type
List[Parameter]

classmethod translate_bool(*val*)

Default behavior for translation

classmethod translate_dict(*val*)

classmethod translate_float(*val*)

Default behavior for translation

classmethod translate_list(*val*)

R

class papermill.translators.**RTranslator**

classmethod assign(*name*, *str_val*)

classmethod comment(*cmt_str*)

classmethod translate_bool(*val*)

Default behavior for translation

classmethod translate_dict(*val*)

classmethod translate_list(*val*)

classmethod translate_none(*val*)

Default behavior for translation

Julia

class papermill.translators.**JuliaTranslator**

classmethod comment(*cmt_str*)

classmethod translate_dict(*val*)

classmethod translate_list(*val*)

classmethod translate_none(*val*)

Default behavior for translation

Scala

class papermill.translators.**ScalaTranslator**

classmethod assign(*name*, *str_val*)

classmethod comment(*cmt_str*)

classmethod translate_dict(*val*)

Translate dicts to scala Maps

classmethod translate_int(*val*)

Default behavior for translation

classmethod translate_list(*val*)

Translate list to scala Seq

Functions

papermill.translators.**translate_parameters**(*kernel_name*, *language*, *parameters*, *comment='Parameters'*)

3.1.4 Input / Output

papermill.iorw

class papermill.iorw.**ABSHandler**

Bases: object

listdir(*path*)

pretty_path(*path*)

read(*path*)

write(*buf*, *path*)

class papermill.iorw.**ADLHandler**

Bases: object

listdir(*path*)

pretty_path(*path*)

read(*path*)

write(*buf*, *path*)

class papermill.iorw.**GCSHandler**

Bases: object

RATE_LIMIT_RETRIES = 3

RETRY_DELAY = 1

RETRY_MAX_DELAY = 4

RETRY_MULTIPLIER = 1

listdir(*path*)

pretty_path(*path*)

read(*path*)

write(*buf*, *path*)

class papermill.iorw.**GithubHandler**

Bases: object

listdir(*path*)

pretty_path(*path*)

read(*path*)

write(*buf*, *path*)

class papermill.iorw.**HDFSHandler**

Bases: object

listdir(*path*)

pretty_path(*path*)

read(*path*)

write(*buf*, *path*)

class papermill.iorw.**HttpHandler**

Bases: object

classmethod listdir(*path*)

classmethod pretty_path(*path*)

classmethod read(*path*)

classmethod write(*buf*, *path*)

class papermill.iorw.**LocalHandler**

Bases: object

cwd(*new_path*)

Sets the cwd during reads and writes

listdir(*path*)

pretty_path(*path*)

read(*path*)

write(*buf*, *path*)

class papermill.iorw.**NoDatesSafeLoader**(*stream*)

Bases: SafeLoader

```
yaml_implicit_resolvers = {'':  [('tag:yaml.org,2002:null', re.compile('^(?:  ~\n
|null|Null|NULL\n | )$', re.VERBOSE))], '!': [('tag:yaml.org,2002:yaml',
re.compile('^(?:!|&|\\*)$'))], '&':  [('tag:yaml.org,2002:yaml',
re.compile('^(?:!|&|\\*)$'))], '*':  [('tag:yaml.org,2002:yaml',
re.compile('^(?:!|&|\\*)$'))], '+':  [('tag:yaml.org,2002:float',
re.compile('^(?:[-+]?(?:[0-9][0-9_]*)\\.[0-9_]*(?:[eE][-+][0-9]+)?\n
|\\.[0-9][0-9_]*(?:[eE][-+][0-9]+)?\n
|[-+]?[0-9][0-9_]*(?::[0-5]?[0-9])+\\.[0-9_]*\n , re.VERBOSE)),
('tag:yaml.org,2002:int', re.compile('^(?:[-+]?0b[0-1_]+\n |[-+]?0[0-7_]+\n
|[-+]?(?:0|[1-9][0-9_]*)\n |[-+]?0x[0-9a-fA-F_]+\n
|[-+]?[1-9][0-9_]*(?::[0-5]?[0-9]), re.VERBOSE))], '-':
[('tag:yaml.org,2002:float',
re.compile('^(?:[-+]?(?:[0-9][0-9_]*)\\.[0-9_]*(?:[eE][-+][0-9]+)?\n
|\\.[0-9][0-9_]*(?:[eE][-+][0-9]+)?\n
|[-+]?[0-9][0-9_]*(?::[0-5]?[0-9])+\\.[0-9_]*\n , re.VERBOSE)),
('tag:yaml.org,2002:int', re.compile('^(?:[-+]?0b[0-1_]+\n |[-+]?0[0-7_]+\n
|[-+]?(?:0|[1-9][0-9_]*)\n |[-+]?0x[0-9a-fA-F_]+\n
|[-+]?[1-9][0-9_]*(?::[0-5]?[0-9]), re.VERBOSE))], '.': [('tag:yaml.org,2002:float',
re.compile('^(?:[-+]?(?:[0-9][0-9_]*)\\.[0-9_]*(?:[eE][-+][0-9]+)?\n
|\\.[0-9][0-9_]*(?:[eE][-+][0-9]+)?\n
|[-+]?[0-9][0-9_]*(?::[0-5]?[0-9])+\\.[0-9_]*\n , re.VERBOSE))], '0':
[('tag:yaml.org,2002:float',
re.compile('^(?:[-+]?(?:[0-9][0-9_]*)\\.[0-9_]*(?:[eE][-+][0-9]+)?\n
|\\.[0-9][0-9_]*(?:[eE][-+][0-9]+)?\n
|[-+]?[0-9][0-9_]*(?::[0-5]?[0-9])+\\.[0-9_]*\n , re.VERBOSE)),
('tag:yaml.org,2002:int', re.compile('^(?:[-+]?0b[0-1_]+\n |[-+]?0[0-7_]+\n
|[-+]?(?:0|[1-9][0-9_]*)\n |[-+]?0x[0-9a-fA-F_]+\n
|[-+]?[1-9][0-9_]*(?::[0-5]?[0-9]), re.VERBOSE))], '1':
[('tag:yaml.org,2002:float',
re.compile('^(?:[-+]?(?:[0-9][0-9_]*)\\.[0-9_]*(?:[eE][-+][0-9]+)?\n
|\\.[0-9][0-9_]*(?:[eE][-+][0-9]+)?\n
|[-+]?[0-9][0-9_]*(?::[0-5]?[0-9])+\\.[0-9_]*\n , re.VERBOSE)),
('tag:yaml.org,2002:int', re.compile('^(?:[-+]?0b[0-1_]+\n |[-+]?0[0-7_]+\n
|[-+]?(?:0|[1-9][0-9_]*)\n |[-+]?0x[0-9a-fA-F_]+\n
|[-+]?[1-9][0-9_]*(?::[0-5]?[0-9]), re.VERBOSE))], '2':
[('tag:yaml.org,2002:float',
re.compile('^(?:[-+]?(?:[0-9][0-9_]*)\\.[0-9_]*(?:[eE][-+][0-9]+)?\n
|\\.[0-9][0-9_]*(?:[eE][-+][0-9]+)?\n
|[-+]?[0-9][0-9_]*(?::[0-5]?[0-9])+\\.[0-9_]*\n , re.VERBOSE)),
('tag:yaml.org,2002:int', re.compile('^(?:[-+]?0b[0-1_]+\n |[-+]?0[0-7_]+\n
|[-+]?(?:0|[1-9][0-9_]*)\n |[-+]?0x[0-9a-fA-F_]+\n
|[-+]?[1-9][0-9_]*(?::[0-5]?[0-9]), re.VERBOSE))], '3':
[('tag:yaml.org,2002:float',
re.compile('^(?:[-+]?(?:[0-9][0-9_]*)\\.[0-9_]*(?:[eE][-+][0-9]+)?\n
|\\.[0-9][0-9_]*(?:[eE][-+][0-9]+)?\n
|[-+]?[0-9][0-9_]*(?::[0-5]?[0-9])+\\.[0-9_]*\n , re.VERBOSE)),
('tag:yaml.org,2002:int', re.compile('^(?:[-+]?0b[0-1_]+\n |[-+]?0[0-7_]+\n
|[-+]?(?:0|[1-9][0-9_]*)\n |[-+]?0x[0-9a-fA-F_]+\n
|[-+]?[1-9][0-9_]*(?::[0-5]?[0-9]), re.VERBOSE))], '4':
[('tag:yaml.org,2002:float',
re.compile('^(?:[-+]?(?:[0-9][0-9_]*)\\.[0-9_]*(?:[eE][-+][0-9]+)?\n
|\\.[0-9][0-9_]*(?:[eE][-+][0-9]+)?\n
|[-+]?[0-9][0-9_]*(?::[0-5]?[0-9])+\\.[0-9_]*\n , re.VERBOSE)),
('tag:yaml.org,2002:int', re.compile('^(?:[-+]?0b[0-1_]+\n |[-+]?0[0-7_]+\n
|[-+]?(?:0|[1-9][0-9_]*)\n |[-+]?0x[0-9a-fA-F_]+\n
|[-+]?[1-9][0-9_]*(?::[0-5]?[0-9]), re.VERBOSE))], '5':
[('tag:yaml.org,2002:float',
re.compile('^(?:[-+]?(?:[0-9][0-9_]*)\\.[0-9_]*(?:[eE][-+][0-9]+)?\n
|\\.[0-9][0-9_]*(?:[eE][-+][0-9]+)?\n
|[-+]?[0-9][0-9_]*(?::[0-5]?[0-9])+\\.[0-9_]*\n , re.VERBOSE)),
```

class papermill.iorw.**NoIOHandler**

Bases: object

Handler for output_path of None - intended to not write anything

listdir(*path*)

pretty_path(*path*)

read(*path*)

write(*buf*, *path*)

class papermill.iorw.**NotebookNodeHandler**

Bases: object

Handler for input_path of nbformat.NotebookNode object

listdir(*path*)

pretty_path(*path*)

read(*path*)

write(*buf*, *path*)

class papermill.iorw.**PapermillIO**

Bases: object

The holder which houses any io system registered with the system. This object is used in a singleton manner to save and load particular named Handler objects for reference externally.

get_handler(*path*, *extensions=None*)

Get I/O Handler based on a notebook path

Parameters

- **path** (*str or nbformat.NotebookNode or None*)
- **extensions** (*list of str, optional*) – Required file extension options for the path (if path is a string), which will log a warning if there is no match. Defaults to None, which does not check for any extensions

Raises

PapermillException – If a valid I/O handler could not be found for the input path:

Return type

I/O Handler

listdir(*path*)

pretty_path(*path*)

read(*path*, *extensions=['.ipynb', '.json']*)

register(*scheme*, *handler*)

register_entry_points()

reset()

write(*buf*, *path*, *extensions=['.ipynb', '.json']*)

class papermill.iorw.**S3Handler**

Bases: object

classmethod listdir(*path*)

classmethod pretty_path(*path*)

classmethod read(*path*)

classmethod write(*buf*, *path*)

class papermill.iorw.**StreamHandler**

Bases: object

Handler for Stdin/Stdout streams

listdir(*path*)

pretty_path(*path*)

read(*path*)

write(*buf*, *path*)

papermill.iorw.**fallback_gs_is_retriable**(*e*)

papermill.iorw.**get_pretty_path**(*path*)

papermill.iorw.**list_notebook_files**(*path*)

Returns a list of all the notebook files in a directory.

papermill.iorw.**load_notebook_node**(*notebook_path*)

Returns a notebook object with papermill metadata loaded from the specified path.

Parameters
notebook_path (*str*) – Path to the notebook file.

Returns
nbformat.NotebookNode

papermill.iorw.**local_file_io_cwd**(*path=None*)

papermill.iorw.**read_yaml_file**(*path*)

Reads a YAML file from the location specified at 'path'.

papermill.iorw.**write_ipynb**(*nb*, *path*)

Saves a notebook object to the specified path. :param nb_node: Notebook object to save. :type nb_node: nbformat.NotebookNode :param notebook_path: Path to save the notebook object to. :type notebook_path: str

3.1.5 Storage

Azure

These modules outline how to interact with Azure data stores, specifically Azure Blob Storage and Azure Data Lakes.

papermill.abs module

Utilities for working with Azure blob storage

class papermill.abs.**AzureBlobStore**

Bases: object

Represents a Blob of storage on Azure

The following are wrapped utilities for Azure storage:

- read
- listdir
- write

listdir(*url*)

Returns a list of the files under the specified path

read(*url*)

Read storage at a given url

write(*buf*, *url*)

Write buffer to storage at a given url

papermill.adl module

Utilities for working with Azure data lake storage

class papermill.adl.**ADL**

Bases: object

Represents an Azure Data Lake

The following are wrapped utilities for Azure storage:

- read

- listdir

- write

listdir(*url*)

Returns a list of the files under the specified path

read(*url*)

Read storage at a given url

write(*buf*, *url*)

Write buffer to storage at a given url

AWS

This module shows how to interact with AWS S3 data stores.

papermill.s3 module

Utilities for working with S3.

class papermill.s3.**Bucket**(*name*, *service=None*)

Bases: object

Represents a Bucket of storage on S3

Parameters

- **name** (*string*) – name of the bucket
- **service** (*string, optional (Default is None)*) – name of a service resource, such as SQS, EC2, etc.

list(*prefix=''*, *delimiter=None*)

Limits a list of Bucket's objects based on prefix and delimiter.

class papermill.s3.**Key**(*bucket*, *name*, *size=None*, *etag=None*, *last_modified=None*, *storage_class=None*, *service=None*)

Bases: object

A key that represents a unique object in an S3 Bucket.

Represents a file or stream.

Parameters

- **bucket** (*object*) – A bucket of S3 storage
- **name** (*string*) – representative name of the bucket
- **size** (*???, optional (Default is None)*)
- **etag** (*???, optional (Default is None)*)
- **last_modified** (*date, optional (Default is None)*)
- **storage_class** (*???, optional (Default is None)*)
- **service** (*string, optional (Default is None)*) – name of a service resource, such as SQS, EC2, etc.

class papermill.s3.**Prefix**(*bucket*, *name*, *service=None*)

Bases: object

Represents a prefix used in an S3 Bucket.

Parameters

- **bucket** (*object*) – A bucket of S3 storage
- **name** (*string*) – name of the bucket
- **service** (*string, optional (Default is None)*) – name of a service resource, such as SQS, EC2, etc.

class papermill.s3.**S3**(*keyname=None*, **args*, ***kwargs*)

Bases: object

Wraps S3.

Parameters

keyname (*TODO*)

The following are wrapped utilities for S3:

- cat
- cp_string
- list
- list_dir
- read

cat(*source*, *buffersize=None*, *memsize=16777216*, *compressed=False*, *encoding='UTF-8'*, *raw=False*)

Returns an iterator for the data in the key or nothing if the key doesn't exist. Decompresses data on the fly (if compressed is True or key ends with .gz) unless raw is True. Pass None for encoding to skip encoding.

cp_string(*source*, *dest*, ***kwargs*)

Copies source string into the destination location.

Parameters

- **source** (*string*) – the string with the content to copy
- **dest** (*string*) – the s3 location

list(*name*, *iterator=False*, ***kwargs*)

Returns a list of the files under the specified path name must be in the form of *s3://bucket/prefix*

Parameters

- **keys** (*optional*) – if True then this will return the actual boto keys for files that are encountered
- **objects** (*optional*) – if True then this will return the actual boto objects for files or prefixes that are encountered
- **delimiter** (*optional*) – if set this
- **iterator** (*optional*) – if True return iterator rather than converting to list object

listdir(*name*, ***kwargs*)

Returns a list of the files under the specified path.

This is different from list as it will only give you files under the current directory, much like ls.

name must be in the form of *s3://bucket/prefix/*

Parameters

- **keys** (*optional*) – if True then this will return the actual boto keys for files that are encountered
- **objects** (*optional*) – if True then this will return the actual boto objects for files or prefixes that are encountered

lock = <unlocked _thread.RLock object owner=0 count=0>

read(*source*, *compressed=False*, *encoding='UTF-8'*)

Iterates over a file in s3 split on newline.

Yields a line in file.

s3_session = (None, None, None)

3.1.6 Utilities

Utils

papermill.utils.**any_tagged_cell**(*nb*, *tag*)

Whether the notebook contains at least one cell tagged `tag`?

Parameters

- **nb** (*nbformat.NotebookNode*) – The notebook to introspect
- **tag** (*str*) – The tag to look for

Returns
Whether the notebook contains a cell tagged `tag`?

Return type
bool

papermill.utils.**chdir**(*path*)

Change working directory to *path* and restore old path on exit.

path can be *None* in which case this is a no-op.

papermill.utils.**find_first_tagged_cell_index**(*nb*, *tag*)

Find the first tagged cell `tag` in the notebook.

Parameters

- **nb** (*nbformat.NotebookNode*) – The notebook to introspect
- **tag** (*str*) – The tag to look for

Returns
Whether the notebook contains a cell tagged `tag`?

Return type
nbformat.NotebookNode

papermill.utils.**merge_kwargs**(*caller_args*, ***callee_args*)

Merge named argument.

Function takes a dictionary of caller arguments and callee arguments as keyword arguments Returns a dictionary with merged arguments. If same argument is in both caller and callee arguments the last one will be taken and warning will be raised.

Parameters

- **caller_args** (*dict*) – Caller arguments
- ****callee_args** – Keyword callee arguments

Returns
args – Merged arguments

Return type
dict

papermill.utils.**nb_kernel_name**(*nb*, *name=None*)

Helper for fetching out the kernel name from a notebook object.

Parameters

- **nb** (*nbformat.NotebookNode*) – The notebook to introspect
- **name** (*str*) – A provided name field

Returns
The name of the kernel or an empty string if none is found

Return type
str

papermill.utils.**nb_language**(*nb*, *language=None*)

Helper for fetching out the programming language from a notebook object.

Parameters

- **nb** (*nbformat.NotebookNode*) – The notebook to introspect
- **language** (*str*) – A provided language field

Returns
The programming language of the notebook

Return type
str

Raises
ValueError – If no notebook language is found or provided

papermill.utils.**remove_args**(*args=None*, ***kwargs*)

Remove arguments from kwargs.

Parameters

- **args** (*list*) – Argument names to remove from kwargs
- ****kwargs** – Arbitrary keyword arguments

Returns
kwargs – New dictionary of arguments

Return type
dict

papermill.utils.**retry**(*num*)

Exceptions

exception papermill.exceptions.**AwsError**

Raised when an AWS Exception is encountered.

exception papermill.exceptions.**FileExistsError**

Raised when a File already exists on S3.

exception papermill.exceptions.**PapermillException**

Raised when an exception is encountered when operating on a notebook.

exception papermill.exceptions.**PapermillExecutionError**(*cell_index*, *exec_count*, *source*, *ename*, *evalue*, *traceback*)

Raised when an exception is encountered in a notebook.

exception papermill.exceptions.**PapermillMissingParameterException**

Raised when a parameter without a value is required to operate on a notebook.

exception papermill.exceptions.**PapermillOptionalDependencyException**

Raised when an exception is encountered when an optional plugin is missing.

exception papermill.exceptions.**PapermillParameterOverwriteWarning**

Callee overwrites caller argument to pass down the stream.

exception papermill.exceptions.**PapermillRateLimitException**

Raised when an io request has been rate limited

exception papermill.exceptions.**PapermillWarning**

Base warning for papermill.

papermill.exceptions.**missing_dependency_generator**(*package*, *dep*)

papermill.exceptions.**missing_environment_variable_generator**(*package*, *env_key*)

papermill.exceptions.**strip_color**(*text*)

Remove most ANSI color and style sequences from a string

Based on https://pypi.org/project/ansicolors/.

Log

Sets up a logger

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

papermill.abs, 36
papermill.adl, 36
papermill.cli, 21
papermill.clientwrap, 26
papermill.engines, 22
papermill.exceptions, 40
papermill.execute, 25
papermill.iorw, 30
papermill.log, 41
papermill.s3, 37
papermill.utils, 39

INDEX

A

ABSHandler (*class in papermill.iorw*), 30
ADL (*class in papermill.adl*), 36
ADLHandler (*class in papermill.iorw*), 30
any_tagged_cell() (*in module papermill.utils*), 39
assign() (*papermill.translators.RTranslator class method*), 29
assign() (*papermill.translators.ScalaTranslator class method*), 30
assign() (*papermill.translators.Translator class method*), 28
autosave_cell() (*papermill.engines.NotebookExecutionManager method*), 23
AwsError, 40
AzureBlobStore (*class in papermill.abs*), 36

B

Bucket (*class in papermill.s3*), 37

C

cat() (*papermill.s3.S3 method*), 38
catch_nb_assignment() (*in module papermill.engines*), 25
cell_complete() (*papermill.engines.NotebookExecutionManager method*), 23
cell_exception() (*papermill.engines.NotebookExecutionManager method*), 24
cell_start() (*papermill.engines.NotebookExecutionManager method*), 24
chdir() (*in module papermill.utils*), 39
cleanup_pbar() (*papermill.engines.NotebookExecutionManager method*), 24
codify() (*papermill.translators.PythonTranslator class method*), 29
codify() (*papermill.translators.Translator class method*), 28
comment() (*papermill.translators.JuliaTranslator class method*), 30
comment() (*papermill.translators.PythonTranslator class method*), 29
comment() (*papermill.translators.RTranslator class method*), 29
comment() (*papermill.translators.ScalaTranslator class method*), 30
comment() (*papermill.translators.Translator class method*), 28
complete_pbar() (*papermill.engines.NotebookExecutionManager method*), 24
COMPLETED (*papermill.engines.NotebookExecutionManager attribute*), 23
cp_string() (*papermill.s3.S3 method*), 38
cwd() (*papermill.iorw.LocalHandler method*), 32

E

Engine (*class in papermill.engines*), 22
execute() (*papermill.clientwrap.PapermillNotebookClient method*), 26
execute_managed_notebook() (*papermill.engines.Engine class method*), 22
execute_managed_notebook() (*papermill.engines.NBClientEngine class method*), 23
execute_notebook() (*in module papermill.execute*), 25
execute_notebook() (*papermill.engines.Engine class method*), 22
execute_notebook_with_engine() (*papermill.engines.PapermillEngines method*), 25

F

FAILED (*papermill.engines.NotebookExecutionManager attribute*), 23
fallback_gs_is_retriable() (*in module papermill.iorw*), 35
FileExistsError, 40
find_first_tagged_cell_index() (*in module papermill.utils*), 39

find_translator() (*papermill.translators.PapermillTranslators method*), 29

G

GCSHandler (*class in papermill.iorw*), 31
get_cell_description() (*papermill.engines.NotebookExecutionManager method*), 24
get_engine() (*papermill.engines.PapermillEngines method*), 25
get_handler() (*papermill.iorw.PapermillIO method*), 34
get_pretty_path() (*in module papermill.iorw*), 35
GithubHandler (*class in papermill.iorw*), 31

H

HDFSHandler (*class in papermill.iorw*), 31
HttpHandler (*class in papermill.iorw*), 31

I

inspect() (*papermill.translators.PythonTranslator class method*), 29
inspect() (*papermill.translators.Translator class method*), 28

J

JuliaTranslator (*class in papermill.translators*), 30

K

Key (*class in papermill.s3*), 37

L

list() (*papermill.s3.Bucket method*), 37
list() (*papermill.s3.S3 method*), 38
list_notebook_files() (*in module papermill.iorw*), 35
listdir() (*papermill.abs.AzureBlobStore method*), 36
listdir() (*papermill.adl.ADL method*), 36
listdir() (*papermill.iorw.ABSHandler method*), 30
listdir() (*papermill.iorw.ADLHandler method*), 30
listdir() (*papermill.iorw.GCSHandler method*), 31
listdir() (*papermill.iorw.GithubHandler method*), 31
listdir() (*papermill.iorw.HDFSHandler method*), 31
listdir() (*papermill.iorw.HttpHandler class method*), 31
listdir() (*papermill.iorw.LocalHandler method*), 32
listdir() (*papermill.iorw.NoIOHandler method*), 34
listdir() (*papermill.iorw.NotebookNodeHandler method*), 34
listdir() (*papermill.iorw.PapermillIO method*), 34
listdir() (*papermill.iorw.S3Handler class method*), 35
listdir() (*papermill.iorw.StreamHandler method*), 35
listdir() (*papermill.s3.S3 method*), 38
load_notebook_node() (*in module papermill.iorw*), 35
local_file_io_cwd() (*in module papermill.iorw*), 35
LocalHandler (*class in papermill.iorw*), 31
lock (*papermill.s3.S3 attribute*), 38
log_output (*papermill.clientwrap.PapermillNotebookClient attribute*), 26
log_output_message() (*papermill.clientwrap.PapermillNotebookClient method*), 26

M

merge_kwargs() (*in module papermill.utils*), 39
missing_dependency_generator() (*in module papermill.exceptions*), 41
missing_environment_variable_generator() (*in module papermill.exceptions*), 41
module
    papermill.abs, 36
    papermill.adl, 36
    papermill.cli, 21
    papermill.clientwrap, 26
    papermill.engines, 22
    papermill.exceptions, 40
    papermill.execute, 25
    papermill.iorw, 30
    papermill.log, 41
    papermill.s3, 37
    papermill.utils, 39

N

nb_kernel_name() (*in module papermill.utils*), 39
nb_kernel_name() (*papermill.engines.Engine class method*), 23
nb_kernel_name() (*papermill.engines.PapermillEngines method*), 25
nb_language() (*in module papermill.utils*), 40
nb_language() (*papermill.engines.Engine class method*), 23
nb_language() (*papermill.engines.PapermillEngines method*), 25
NBClientEngine (*class in papermill.engines*), 23
NoDatesSafeLoader (*class in papermill.iorw*), 32
NoIOHandler (*class in papermill.iorw*), 34
notebook_complete() (*papermill.engines.NotebookExecutionManager method*), 24
notebook_start() (*papermill.engines.NotebookExecutionManager method*), 24
NotebookExecutionManager (*class in papermill.engines*), 23
NotebookNodeHandler (*class in papermill.iorw*), 34

now() (*papermill.engines.NotebookExecutionManager method*), 24

P

papermill.abs
    module, 36
papermill.adl
    module, 36
papermill.cli
    module, 21
papermill.clientwrap
    module, 26
papermill.engines
    module, 22
papermill.exceptions
    module, 40
papermill.execute
    module, 25
papermill.iorw
    module, 30
papermill.log
    module, 41
papermill.s3
    module, 37
papermill.utils
    module, 39
papermill_execute_cells() (*papermill.clientwrap.PapermillNotebookClient method*), 27
PapermillEngines (*class in papermill.engines*), 24
PapermillException, 40
PapermillExecutionError, 40
PapermillIO (*class in papermill.iorw*), 34
PapermillMissingParameterException, 40
PapermillNotebookClient (*class in papermill.clientwrap*), 26
PapermillOptionalDependencyException, 40
PapermillParameterOverwriteWarning, 40
PapermillRateLimitException, 41
PapermillTranslators (*class in papermill.translators*), 29
PapermillWarning, 41
PARAMETER_PATTERN (*papermill.translators.PythonTranslator attribute*), 29
PENDING (*papermill.engines.NotebookExecutionManager attribute*), 23
Prefix (*class in papermill.s3*), 37
prepare_notebook_metadata() (*in module papermill.execute*), 26
pretty_path() (*papermill.iorw.ABSHandler method*), 30
pretty_path() (*papermill.iorw.ADLHandler method*), 31
pretty_path() (*papermill.iorw.GCSHandler method*), 31
pretty_path() (*papermill.iorw.GithubHandler method*), 31
pretty_path() (*papermill.iorw.HDFSHandler method*), 31
pretty_path() (*papermill.iorw.HttpHandler class method*), 31
pretty_path() (*papermill.iorw.LocalHandler method*), 32
pretty_path() (*papermill.iorw.NoIOHandler method*), 34
pretty_path() (*papermill.iorw.NotebookNodeHandler method*), 34
pretty_path() (*papermill.iorw.PapermillIO method*), 34
pretty_path() (*papermill.iorw.S3Handler class method*), 35
pretty_path() (*papermill.iorw.StreamHandler method*), 35
print_papermill_version() (*in module papermill.cli*), 21
process_message() (*papermill.clientwrap.PapermillNotebookClient method*), 27
PythonTranslator (*class in papermill.translators*), 29

R

raise_for_execution_errors() (*in module papermill.execute*), 26
RATE_LIMIT_RETRIES (*papermill.iorw.GCSHandler attribute*), 31
read() (*papermill.abs.AzureBlobStore method*), 36
read() (*papermill.adl.ADL method*), 36
read() (*papermill.iorw.ABSHandler method*), 30
read() (*papermill.iorw.ADLHandler method*), 31
read() (*papermill.iorw.GCSHandler method*), 31
read() (*papermill.iorw.GithubHandler method*), 31
read() (*papermill.iorw.HDFSHandler method*), 31
read() (*papermill.iorw.HttpHandler class method*), 31
read() (*papermill.iorw.LocalHandler method*), 32
read() (*papermill.iorw.NoIOHandler method*), 34
read() (*papermill.iorw.NotebookNodeHandler method*), 34
read() (*papermill.iorw.PapermillIO method*), 34
read() (*papermill.iorw.S3Handler class method*), 35
read() (*papermill.iorw.StreamHandler method*), 35
read() (*papermill.s3.S3 method*), 38
read_yaml_file() (*in module papermill.iorw*), 35
register() (*papermill.engines.PapermillEngines method*), 25
register() (*papermill.iorw.PapermillIO method*), 34
register() (*papermill.translators.PapermillTranslators method*), 29

register_entry_points() (*papermill.engines.PapermillEngines method*), 25
register_entry_points() (*papermill.iorw.PapermillIO method*), 34
remove_args() (*in module papermill.utils*), 40
remove_error_markers() (*in module papermill.execute*), 26
reset() (*papermill.iorw.PapermillIO method*), 34
retry() (*in module papermill.utils*), 40
RETRY_DELAY (*papermill.iorw.GCSHandler attribute*), 31
RETRY_MAX_DELAY (*papermill.iorw.GCSHandler attribute*), 31
RETRY_MULTIPLIER (*papermill.iorw.GCSHandler attribute*), 31
RTranslator (*class in papermill.translators*), 29
RUNNING (*papermill.engines.NotebookExecutionManager attribute*), 23

S

S3 (*class in papermill.s3*), 37
s3_session (*papermill.s3.S3 attribute*), 38
S3Handler (*class in papermill.iorw*), 35
save() (*papermill.engines.NotebookExecutionManager method*), 24
ScalaTranslator (*class in papermill.translators*), 30
set_timer() (*papermill.engines.NotebookExecutionManager method*), 24
stderr_file (*papermill.clientwrap.PapermillNotebookClient attribute*), 27
stdout_file (*papermill.clientwrap.PapermillNotebookClient attribute*), 27
StreamHandler (*class in papermill.iorw*), 35
strip_color() (*in module papermill.exceptions*), 41

T

translate() (*papermill.translators.Translator class method*), 28
translate_bool() (*papermill.translators.PythonTranslator class method*), 29
translate_bool() (*papermill.translators.RTranslator class method*), 29
translate_bool() (*papermill.translators.Translator class method*), 28
translate_dict() (*papermill.translators.JuliaTranslator class method*), 30
translate_dict() (*papermill.translators.PythonTranslator class method*), 29
translate_dict() (*papermill.translators.RTranslator class method*), 29
translate_dict() (*papermill.translators.ScalaTranslator class method*), 30
translate_dict() (*papermill.translators.Translator class method*), 28
translate_escaped_str() (*papermill.translators.Translator class method*), 28
translate_float() (*papermill.translators.PythonTranslator class method*), 29
translate_float() (*papermill.translators.Translator class method*), 28
translate_int() (*papermill.translators.ScalaTranslator class method*), 30
translate_int() (*papermill.translators.Translator class method*), 28
translate_list() (*papermill.translators.JuliaTranslator class method*), 30
translate_list() (*papermill.translators.PythonTranslator class method*), 29
translate_list() (*papermill.translators.RTranslator class method*), 29
translate_list() (*papermill.translators.ScalaTranslator class method*), 30
translate_list() (*papermill.translators.Translator class method*), 28
translate_none() (*papermill.translators.JuliaTranslator class method*), 30
translate_none() (*papermill.translators.RTranslator class method*), 30
translate_none() (*papermill.translators.Translator class method*), 28
translate_parameters() (*in module papermill.translators*), 30
translate_raw_str() (*papermill.translators.Translator class method*), 28
translate_str() (*papermill.translators.Translator class method*), 28
Translator (*class in papermill.translators*), 28

W

write() (*papermill.abs.AzureBlobStore method*), 36
write() (*papermill.adl.ADL method*), 36
write() (*papermill.iorw.ABSHandler method*), 30
write() (*papermill.iorw.ADLHandler method*), 31
write() (*papermill.iorw.GCSHandler method*), 31
write() (*papermill.iorw.GithubHandler method*), 31

write() (*papermill.iorw.HDFSHandler method*), 31
write() (*papermill.iorw.HttpHandler class method*), 31
write() (*papermill.iorw.LocalHandler method*), 32
write() (*papermill.iorw.NoIOHandler method*), 34
write() (*papermill.iorw.NotebookNodeHandler method*), 34
write() (*papermill.iorw.PapermillIO method*), 34
write() (*papermill.iorw.S3Handler class method*), 35
write() (*papermill.iorw.StreamHandler method*), 35
write_ipynb() (*in module papermill.iorw*), 35

Y

yaml_implicit_resolvers (*papermill.iorw.NoDatesSafeLoader attribute*), 32