
Jupyter Notebook Documentation

Release 1.1.0.dev0

<https://jupyter.org>

Aug 22, 2023

CONTENTS

1	User Documentation	3
2	Configuration	77
3	Contribute	133

Getting Started

USER DOCUMENTATION

1.1 Jupyter NbClassic

1.1.1 Introduction

The nbclassic package is the implementation of the classic Jupyter Notebook 6 as a Jupyter Server extension. As proposed in the accepted [Jupyter Enhancement Proposal #79](#) the development of this package is due to plans being carried out to create a Notebook 7, that is based on the modern JupyterLab code-base.

While Notebook 7 provides a user experience equivalent to that of the classic Notebook 6, the technology stack used is incompatible with that which many Jupyter Notebook users have developed their Jupyter Notebook extensions with. Users may find themselves in need of a way to continue using the Jupyter Notebook 6 tech-stack as they transition to using the Jupyter Notebook 7. The NbClassic package intends to address that need.

You can read more about the migration impacts and coexistence of NbClassic and Notebook package on the [NbClassic and Notebook migration plan](#) page.

1.1.2 NbClassic Usage

Installation

Installing from PyPI: `pip install nbclassic` This will automatically enable the extension in Jupyter Server.

Launch directly: `jupyter nbclassic`

Alternatively, you can run Jupyter Server and visit the */tree* endpoint: `jupyter server`

Configuration

To create a `jupyter_nbclassic_config.py` file in the `.jupyter` directory you can use the following command:

```
$ jupyter nbclassic --generate-config
```

Options

You can view a list of the available options by typing:

```
$ jupyter nbclassic --help
```

1.1.3 NbClassic Development

`Nbclassic` is the package that holds the UI components of the classic Jupyter Notebook 6 and serves this UI through the server endpoints provided by Jupyter Server.

Jupyter Server Extensions - [Authoring a basic server extension](#)

The jupyter/notebook Repository: The original `jupyter/notebook` GitHub repository now holds the codebase for the new Jupyter Notebook (version 7).

1.1.4 NbClassic Timeline

As proposed in the [JEP #79](#), the `nbclassic` package will continue to be supported with critical security fixes in the transition period as users move to using the more modern Jupyter Notebook 7.

Porting Notebook 6 Extensions: Work being done in parallel. You can find a helpful list of classical Notebook extensions and corresponding Jupyterlab extensions if available at the [Jupyterlab-contrib website](#).

1.2 The Jupyter Notebook

- [NbClassic Installation](#)
- [Starting the Notebook](#)

1.2.1 Introduction

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. Jupyter NbClassic combines two components:

A web application: a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.

Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

See also:

See the [installation guide](#) on how to install `nbclassic` and its dependencies.

Main features of the web application

- In-browser editing for code, with automatic syntax highlighting, indentation, and tab completion/introspection.
- The ability to execute code from the browser, with the results of computations attached to the code which generated them.
- Displaying the result of computation using rich media representations, such as HTML, LaTeX, PNG, SVG, etc. For example, publication-quality figures rendered by the `matplotlib` library, can be included inline.
- In-browser editing for rich text using the `Markdown` markup language, which can provide commentary for the code, is not limited to plain text.
- The ability to easily include mathematical notation within markdown cells using LaTeX, and rendered natively by `MathJax`.

Notebook documents

Notebook documents contains the inputs and outputs of a interactive session as well as additional text that accompanies the code but is not meant for execution. In this way, notebook files can serve as a complete computational record of a session, interleaving executable code with explanatory text, mathematics, and rich representations of resulting objects. These documents are internally `JSON` files and are saved with the `.ipynb` extension. Since `JSON` is a plain text format, they can be version-controlled and shared with colleagues.

Notebooks may be exported to a range of static formats, including HTML (for example, for blog posts), reStructuredText, LaTeX, PDF, and slide shows, via the `nbconvert` command.

Furthermore, any `.ipynb` notebook document available from a public URL can be shared via the Jupyter Notebook Viewer `<nbviewer>`. This service loads the notebook document from the URL and renders it as a static web page. The results may thus be shared with a colleague, or as a public blog post, without other users needing to install the Jupyter notebook themselves. In effect, `nbviewer` is simply `nbconvert` as a web service, so you can do your own static conversions with `nbconvert`, without relying on `nbviewer`.

See also:

[Details on the notebook JSON file format](#)

Notebooks and privacy

Because you use Jupyter in a web browser, some people are understandably concerned about using it with sensitive data. However, if you followed the standard [install instructions](#), Jupyter is actually running on your own computer. If the URL in the address bar starts with `http://localhost:` or `http://127.0.0.1:`, it's your computer acting as the server. Jupyter doesn't send your data anywhere else—and as it's open source, other people can check that we're being honest about this.

You can also use Jupyter remotely: your company or university might run the server for you, for instance. If you want to work with sensitive data in those cases, talk to your IT or data protection staff about it.

We aim to ensure that other pages in your browser or other users on the same computer can't access your notebook server. See `server_security` for more about this.

1.2.2 Starting the NbClassic server

You can start running a nbclassic server from the command line using the following command:

```
jupyter nbclassic
```

This will print some information about the nbclassic server in your console, and open a web browser to the URL of the web application (by default, `http://127.0.0.1:8888`).

The landing page of the Jupyter nbclassic web application, the **dashboard**, shows the notebooks currently available in the notebook directory (by default, the directory from which the nbclassic server was started).

You can create new notebooks from the dashboard with the **New Notebook** button, or open existing ones by clicking on their name. You can also drag and drop `.ipynb` notebooks and standard `.py` Python source code files into the notebook list area.

When starting a nbclassic server from the command line, you can also open a particular notebook directly, bypassing the dashboard, with `jupyter nbclassic my_notebook.ipynb`. The `.ipynb` extension is assumed if no extension is given.

When you are inside an open notebook, the *File | Open...* menu option will open the dashboard in a new browser tab, to allow you to open another notebook from the notebook directory or to create a new notebook.

Note: You can start more than one nbclassic server at the same time, if you want to work on notebooks in different directories. By default the first nbclassic server starts on port 8888, and later nbclassic servers search for ports near that one. You can also manually specify the port with the `--port` option.

Creating a new notebook document

A new notebook may be created at any time, either from the dashboard, or using the *File → New* menu option from within an active notebook. The new notebook is created within the same directory and will open in a new browser tab. It will also be reflected as a new entry in the notebook list on the dashboard.

Opening notebooks

An open notebook has **exactly one** interactive session connected to a kernel, which will execute code sent by the user and communicate back results. This kernel remains active if the web browser window is closed, and reopening the same notebook from the dashboard will reconnect the web application to the same kernel. In the dashboard, notebooks with an active kernel have a **Shutdown** button next to them, whereas notebooks without an active kernel have a **Delete** button in its place.

Other clients may connect to the same kernel. When each kernel is started, the nbclassic server prints to the terminal a message like this:

```
[ServerApp] Kernel started: 87f7d2c0-13e3-43df-8bb8-1bd37aaf3373
```

This long string is the kernel's ID which is sufficient for getting the information necessary to connect to the kernel. If the notebook uses the IPython kernel, you can also see this connection data by running the `%connect_info` magic, which will print the same ID information along with other details.

You can then, for example, manually start a Qt console connected to the *same* kernel from the command line, by passing a portion of the ID:

```
$ jupyter qtconsole --existing 87f7d2c0
```

Without an ID, `--existing` will connect to the most recently started kernel.

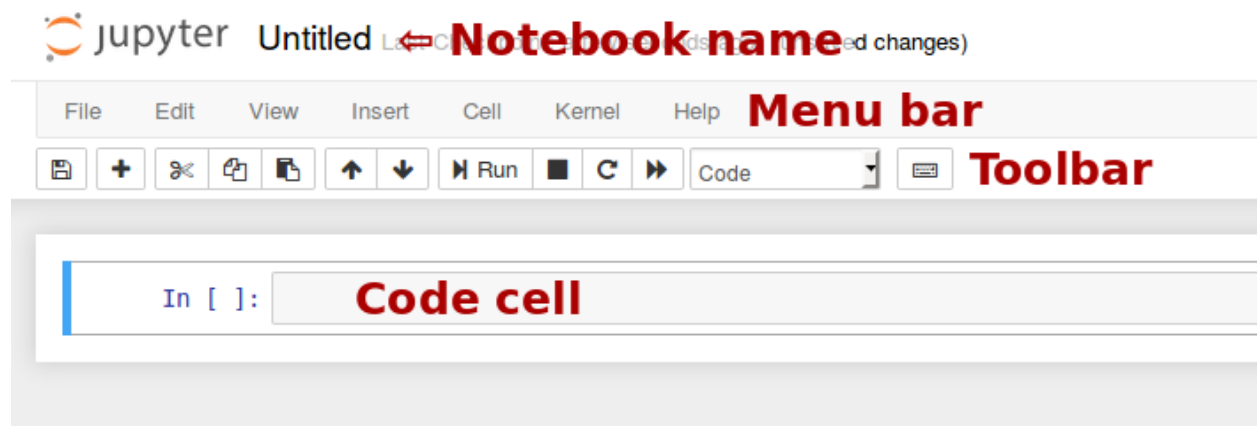
With the IPython kernel, you can also run the `%qtconsole` [magic](#) in the notebook to open a Qt console connected to the same kernel.

See also:

[Decoupled two-process model](#)

1.2.3 Notebook user interface

When you create a new notebook document, you will be presented with the **notebook name**, a **menu bar**, a **toolbar** and an empty **code cell**.



Notebook name: The name displayed at the top of the page, next to the Jupyter logo, reflects the name of the `.ipynb` file. Clicking on the notebook name brings up a dialog which allows you to rename it. Thus, renaming a notebook from “Untitled0” to “My first notebook” in the browser, renames the `Untitled0.ipynb` file to `My first notebook.ipynb`.

Menu bar: The menu bar presents different options that may be used to manipulate the way the notebook functions.

Toolbar: The tool bar gives a quick way of performing the most-used operations within the notebook, by clicking on an icon.

Code cell: the default type of cell; read on for an explanation of cells.

1.2.4 Structure of a notebook document

The notebook consists of a sequence of cells. A cell is a multiline text input field, and its contents can be executed by using `Shift-Enter`, or by clicking either the “Play” button the toolbar, or *Cell*, *Run* in the menu bar. The execution behavior of a cell is determined by the cell’s type. There are three types of cells: **code cells**, **markdown cells**, and **raw cells**. Every cell starts off being a **code cell**, but its type can be changed by using a drop-down on the toolbar (which will be “Code”, initially), or via [keyboard shortcuts](#).

For more information on the different things you can do in a notebook, see the [collection of examples](#).

Code cells

A *code cell* allows you to edit and write new code, with full syntax highlighting and tab completion. The programming language you use depends on the *kernel*, and the default kernel (IPython) runs Python code.

When a code cell is executed, code that it contains is sent to the kernel associated with the notebook. The results that are returned from this computation are then displayed in the notebook as the cell's *output*. The output is not limited to text, with many other possible forms of output are also possible, including `matplotlib` figures and HTML tables (as used, for example, in the `pandas` data analysis package). This is known as IPython's *rich display* capability.

See also:

[Rich Output example notebook](#)

Markdown cells

You can document the computational process in a literate way, alternating descriptive text with code, using *rich text*. In IPython this is accomplished by marking up text with the Markdown language. The corresponding cells are called *Markdown cells*. The Markdown language provides a simple way to perform this text markup, that is, to specify which parts of the text should be emphasized (italics), bold, form lists, etc.

If you want to provide structure for your document, you can use markdown headings. Markdown headings consist of 1 to 6 hash # signs # followed by a space and the title of your section. The markdown heading will be converted to a clickable link for a section of the notebook. It is also used as a hint when exporting to other document formats, like PDF.

When a Markdown cell is executed, the Markdown code is converted into the corresponding formatted rich text. Markdown allows arbitrary HTML code for formatting.

Within Markdown cells, you can also include *mathematics* in a straightforward way, using standard LaTeX notation: `...` for inline mathematics and `$$...$$` for displayed mathematics. When the Markdown cell is executed, the LaTeX portions are automatically rendered in the HTML output as equations with high quality typography. This is made possible by [MathJax](#), which supports a [large subset](#) of LaTeX functionality

Standard mathematics environments defined by LaTeX and AMS-LaTeX (the `amsmath` package) also work, such as `\begin{equation}...\end{equation}`, and `\begin{align}...\end{align}`. New LaTeX macros may be defined using standard methods, such as `\newcommand`, by placing them anywhere *between math delimiters* in a Markdown cell. These definitions are then available throughout the rest of the IPython session.

See also:

[Working with Markdown Cells example notebook](#)

Raw cells

Raw cells provide a place in which you can write *output* directly. Raw cells are not evaluated by the notebook. When passed through `nbconvert`, raw cells arrive in the destination format unmodified. For example, you can type full LaTeX into a raw cell, which will only be rendered by LaTeX after conversion by `nbconvert`.

1.2.5 Basic workflow

The normal workflow in a notebook is, then, quite similar to a standard IPython session, with the difference that you can edit cells in-place multiple times until you obtain the desired results, rather than having to rerun separate scripts with the `%run` magic command.

Typically, you will work on a computational problem in pieces, organizing related ideas into cells and moving forward once previous parts work correctly. This is much more convenient for interactive exploration than breaking up a computation into scripts that must be executed together, as was previously necessary, especially if parts of them take a long time to run.

To interrupt a calculation which is taking too long, use the *Kernel, Interrupt* menu option, or the `i, i` keyboard shortcut. Similarly, to restart the whole computational process, use the *Kernel, Restart* menu option or `0, 0` shortcut.

A notebook may be downloaded as a `.ipynb` file or converted to a number of other formats using the menu option *File, Download as*.

See also:

[Running Code in the Jupyter Notebook](#) example notebook

[Notebook Basics](#) example notebook

Keyboard shortcuts

All actions in the notebook can be performed with the mouse, but keyboard shortcuts are also available for the most common ones. The essential shortcuts to remember are the following:

- **Shift-Enter: run cell**
Execute the current cell, show any output, and jump to the next cell below. If Shift-Enter is invoked on the last cell, it makes a new cell below. This is equivalent to clicking the *Cell, Run* menu item, or the Play button in the toolbar.
- **Esc: Command mode**
In command mode, you can navigate around the notebook using keyboard shortcuts.
- **Enter: Edit mode**
In edit mode, you can edit text in cells.

For the full list of available shortcuts, click *Help, Keyboard Shortcuts* in the notebook menus.

1.2.6 Plotting

One major feature of the Jupyter notebook is the ability to display plots that are the output of running code cells. The IPython kernel is designed to work seamlessly with the `matplotlib` plotting library to provide this functionality. Specific plotting library integration is a feature of the kernel.

1.2.7 Installing kernels

For information on how to install a Python kernel, refer to the [IPython install page](#).

The Jupyter wiki has a long list of [Kernels for other languages](#). They usually come with instructions on how to make the kernel available in the notebook.

1.2.8 Trusting Notebooks

To prevent untrusted code from executing on users' behalf when notebooks open, we store a signature of each trusted notebook. The nbclassic server verifies this signature when a notebook is opened. If no matching signature is found, Javascript and HTML output will not be displayed until they are regenerated by re-executing the cells.

Any notebook that you have fully executed yourself will be considered trusted, and its HTML and Javascript output will be displayed on load.

If you need to see HTML or Javascript output without re-executing, and you are sure the notebook is not malicious, you can tell Jupyter to trust it at the command-line with:

```
$ jupyter trust mynotebook.ipynb
```

See [Security in notebook documents](#) for more details about the trust mechanism.

1.2.9 Browser Compatibility

Jupyter NbClassic aims to support the latest versions of these browsers:

- Chrome
- Safari
- Firefox

Up to date versions of Opera and Edge may also work, but if they don't, please use one of the supported browsers.

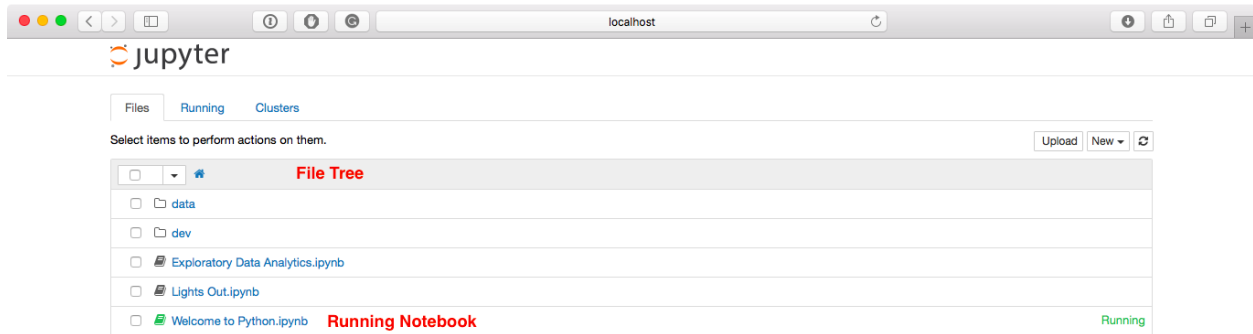
Using Safari with HTTPS and an untrusted certificate is known to not work (websockets will fail).

1.3 User interface components

When opening bug reports or sending emails to the Jupyter mailing list, it is useful to know the names of different UI components so that other developers and users have an easier time helping you diagnose your problems. This section will familiarize you with the names of UI elements within the Notebook and the different Notebook modes.

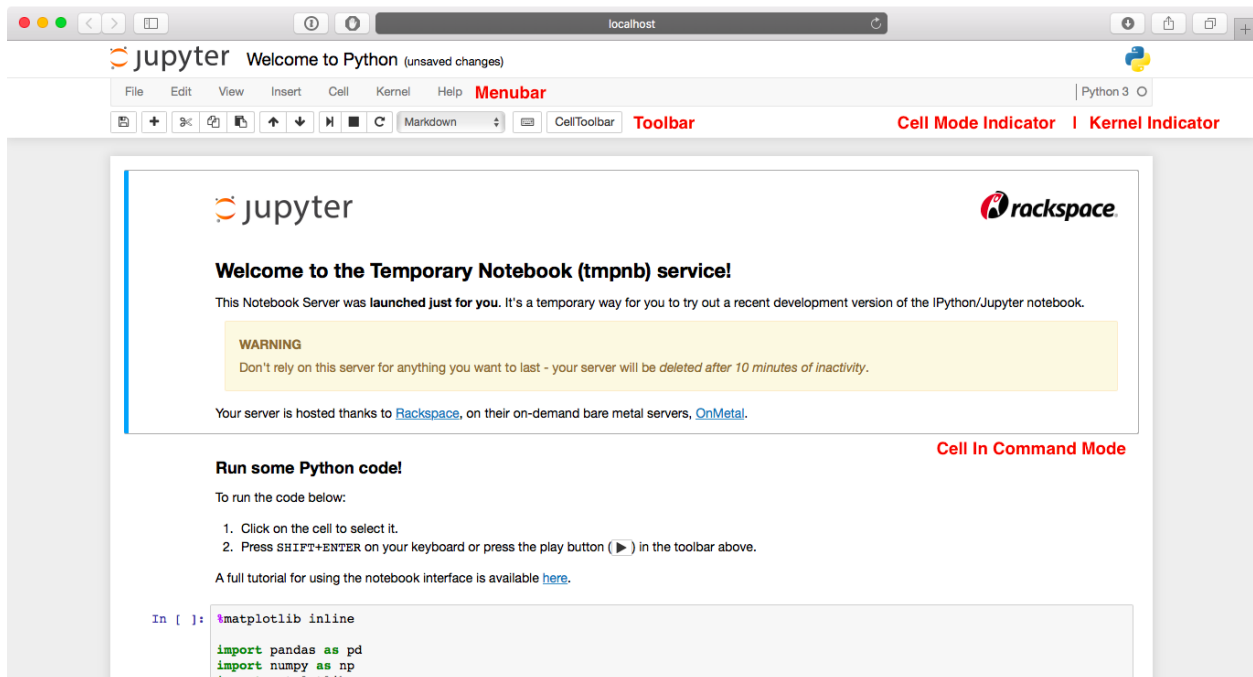
1.3.1 Notebook Dashboard

When you launch `jupyter nbclassic` the first page that you encounter is the Notebook Dashboard.



1.3.2 Notebook Editor

Once you've selected a Notebook to edit, the Notebook will open in the Notebook Editor.

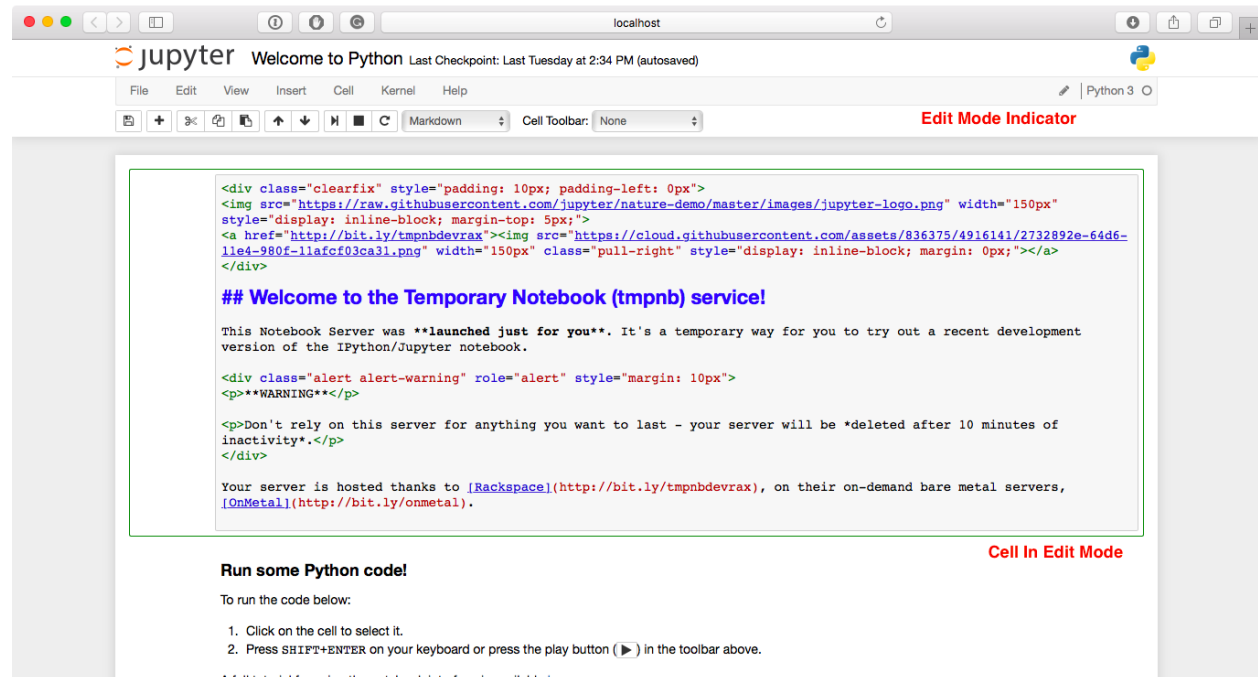


1.3.3 Interactive User Interface Tour of the Notebook

If you would like to learn more about the specific elements within the Notebook Editor, you can go through the user interface tour by selecting *Help* in the menubar then selecting *User Interface Tour*.

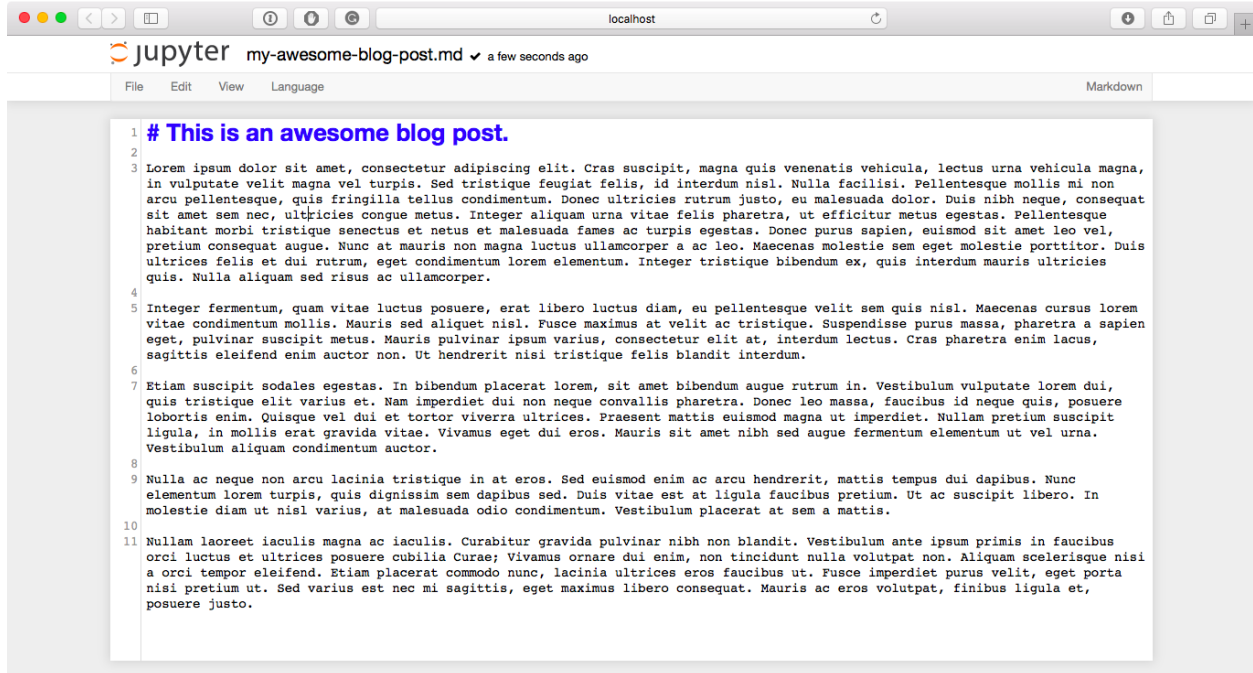
Edit Mode and Notebook Editor

When a cell is in edit mode, the Cell Mode Indicator will change to reflect the cell's state. This state is indicated by a small pencil icon on the top right of the interface. When the cell is in command mode, there is no icon in that location.



1.3.4 File Editor

Now let's say that you've chosen to open a Markdown file instead of a Notebook file whilst in the Notebook Dashboard. If so, the file will be opened in the File Editor.



1.4 Notebook Examples

The pages in this section are all converted notebook files. You can also [view these notebooks on nbviewer](#).

1.4.1 What is the Jupyter Notebook?

Introduction

The Jupyter Notebook is an **interactive computing environment** that enables users to author notebook documents that include: - Live code - Interactive widgets - Plots - Narrative text - Equations - Images - Video

These documents provide a **complete and self-contained record of a computation** that can be converted to various formats and shared with others using email, [Dropbox](#), version control systems (like [git/GitHub](#)) or [nbviewer.jupyter.org](#).

Components

The Jupyter Notebook combines three components:

- **The notebook web application:** An interactive web application for writing and running code interactively and authoring notebook documents.
- **Kernels:** Separate processes started by the notebook web application that runs users' code in a given language and returns output back to the notebook web application. The kernel also handles things like computations for interactive widgets, tab completion and introspection.
- **Notebook documents:** Self-contained documents that contain a representation of all content visible in the notebook web application, including inputs and outputs of the computations, narrative text, equations, images, and rich media representations of objects. Each notebook document has its own kernel.

Notebook web application

The notebook web application enables users to:

- **Edit code in the browser**, with automatic syntax highlighting, indentation, and tab completion/introspection.
- **Run code from the browser**, with the results of computations attached to the code which generated them.
- See the results of computations with **rich media representations**, such as HTML, LaTeX, PNG, SVG, PDF, etc.
- Create and use **interactive JavaScript widgets**, which bind interactive user interface controls and visualizations to reactive kernel side computations.
- Author **narrative text** using the [Markdown](#) markup language.
- Include mathematical equations using **LaTeX syntax in Markdown**, which are rendered in-browser by [MathJax](#).

Kernels

Through Jupyter’s kernel and messaging architecture, the Notebook allows code to be run in a range of different programming languages. For each notebook document that a user opens, the web application starts a kernel that runs the code for that notebook. Each kernel is capable of running code in a single programming language and there are kernels available in the following languages:

- Python(<https://github.com/ipython/ipython>)
- Julia (<https://github.com/JuliaLang/IJulia.jl>)
- R (<https://github.com/IRkernel/IRkernel>)
- Ruby (<https://github.com/minrk/iruby>)
- Haskell (<https://github.com/gibiansky/IHaskell>)
- Scala (<https://github.com/Bridgewater/scala-notebook>)
- node.js (<https://gist.github.com/Carreau/4279371>)
- Go (<https://github.com/takluyver/igo>)

The default kernel runs Python code. The notebook provides a simple way for users to pick which of these kernels is used for a given notebook.

Each of these kernels communicate with the notebook web application and web browser using a JSON over ZeroMQ/WebSockets message protocol that is described [here](#). Most users don’t need to know about these details, but it helps to understand that “kernels run code.”

Notebook documents

Notebook documents contain the **inputs and outputs** of an interactive session as well as **narrative text** that accompanies the code but is not meant for execution. **Rich output** generated by running code, including HTML, images, video, and plots, is embedded in the notebook, which makes it a complete and self-contained record of a computation.

When you run the notebook web application on your computer, notebook documents are just **files on your local filesystem with a .ipynb extension**. This allows you to use familiar workflows for organizing your notebooks into folders and sharing them with others.

Notebooks consist of a **linear sequence of cells**. There are three basic cell types:

- **Code cells:** Input and output of live code that is run in the kernel
- **Markdown cells:** Narrative text with embedded LaTeX equations

- **Raw cells:** Unformatted text that is included, without modification, when notebooks are converted to different formats using nbconvert

Internally, notebook documents are **JSON data** with **binary values** `base64` encoded. This allows them to be **read and manipulated programmatically** by any programming language. Because JSON is a text format, notebook documents are version control friendly.

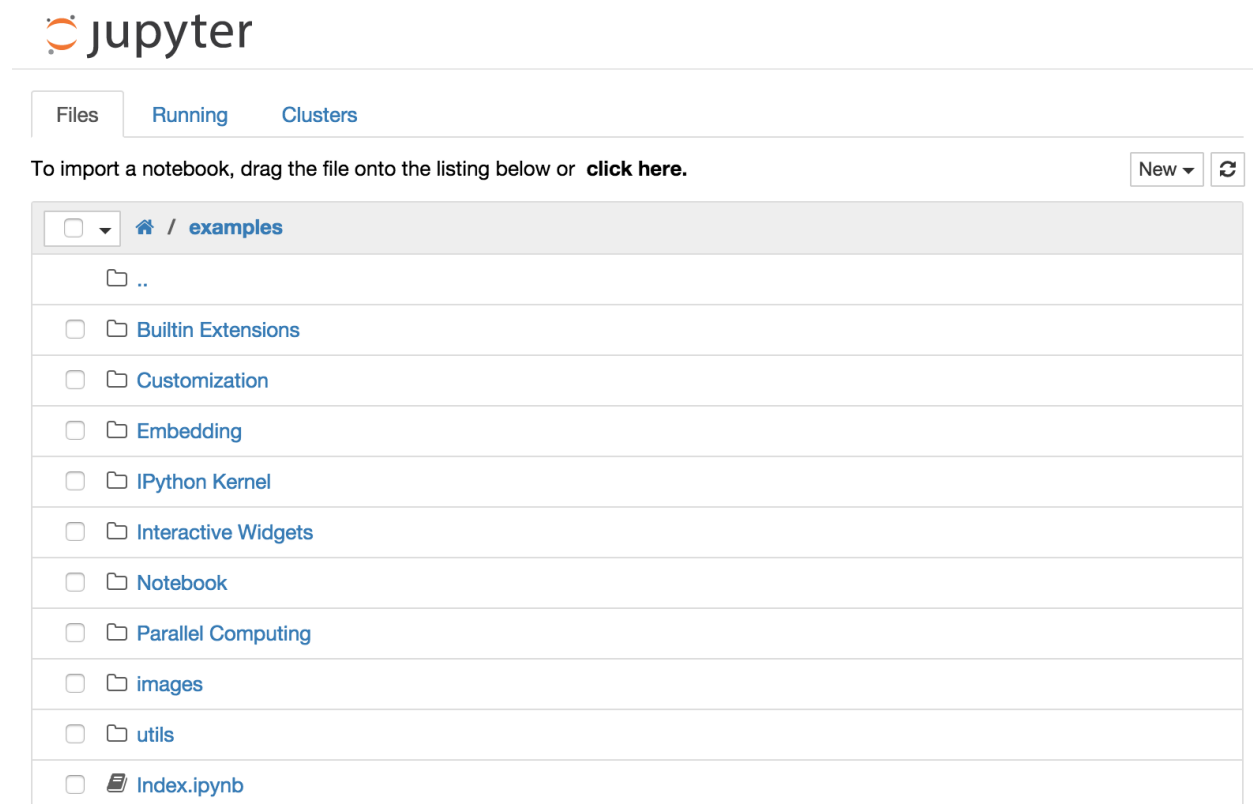
Notebooks can be exported to different static formats including HTML, reStructuredText, LaTeX, PDF, and slide shows (`reveal.js`) using Jupyter’s nbconvert utility.

Furthermore, any notebook document available from a **public URL or on GitHub can be shared** via `nbviewer`. This service loads the notebook document from the URL and renders it as a static web page. The resulting web page may thus be shared with others **without their needing to install the Jupyter Notebook**.

1.4.2 Notebook Basics

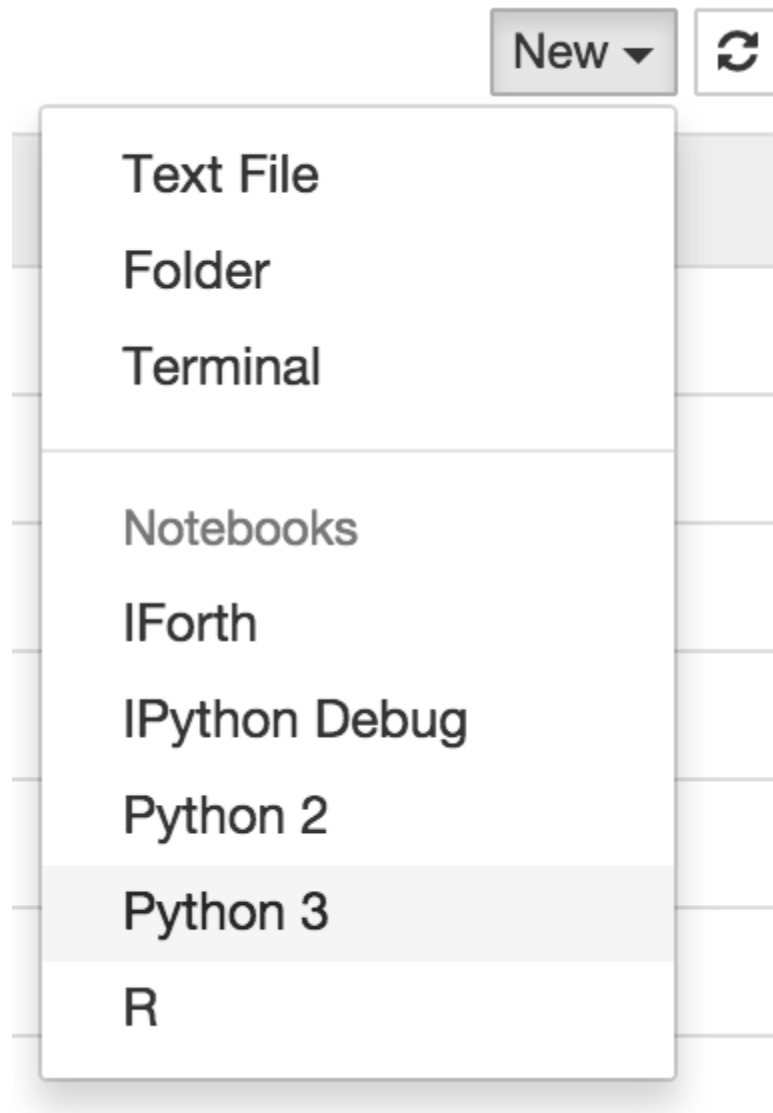
The Notebook dashboard

When you first start the notebook server, your browser will open to the notebook dashboard. The dashboard serves as a home page for the notebook. Its main purpose is to display the notebooks and files in the current directory. For example, here is a screenshot of the dashboard page for the `examples` directory in the Jupyter repository:




The top of the notebook list displays clickable breadcrumbs of the current directory. By clicking on these breadcrumbs or on sub-directories in the notebook list, you can navigate your file system.

To create a new notebook, click on the “New” button at the top of the list and select a kernel from the dropdown (as seen below). Which kernels are listed depend on what’s installed on the server. Some of the kernels in the screenshot below may not exist as an option to you.



Notebooks and files can be uploaded to the current directory by dragging a notebook file onto the notebook list or by the “click here” text above the list.

The notebook list shows green “Running” text and a green notebook icon next to running notebooks (as seen below). Notebooks remain running until you explicitly shut them down; closing the notebook’s page is not sufficient.

	 / examples
	..
<input type="checkbox"/>	 Builtin Extensions
<input type="checkbox"/>	 Customization
<input type="checkbox"/>	 Embedding
<input type="checkbox"/>	 IPython Kernel
<input type="checkbox"/>	 Interactive Widgets
<input type="checkbox"/>	 Notebook
<input type="checkbox"/>	 Parallel Computing
<input type="checkbox"/>	 images
<input type="checkbox"/>	 utils
<input checked="" type="checkbox"/>	 Index.ipynb Running

To shutdown, delete, duplicate, or rename a notebook check the checkbox next to it and an array of controls will appear at the top of the notebook list (as seen below). You can also use the same operations on directories and files when applicable.



To see all of your running notebooks along with their directories, click on the “Running” tab:



Files Running Clusters

Currently running Jupyter processes



Terminals ▾

There are no terminals running.

Notebooks ▾

examples/Notebook/Index.ipynb	Shutdown
examples/Notebook/What is the IPython Notebook.ipynb	Shutdown
examples/Notebook/Running the Notebook Server.ipynb	Shutdown
examples/Notebook/Notebook Basics.ipynb	Shutdown
examples/Notebook/Running Code.ipynb	Shutdown
examples/Notebook/Working With Markdown Cells.ipynb	Shutdown
examples/Notebook/Custom Keyboard Shortcuts.ipynb	Shutdown
examples/Notebook/JavaScript Notebook Extensions.ipynb	Shutdown
examples/Notebook/Notebook Security.ipynb	Shutdown

This view provides a convenient way to track notebooks that you start as you navigate the file system in a long running notebook server.

Overview of the Notebook UI

If you create a new notebook or open an existing one, you will be taken to the notebook user interface (UI). This UI allows you to run code and author notebook documents interactively. The notebook UI has the following main areas:

- Menu
- Toolbar
- Notebook area and cells

The notebook has an interactive tour of these elements that can be started in the “Help:User Interface Tour” menu item.

Modal editor

Starting with IPython 2.0, the Jupyter Notebook has a modal user interface. This means that the keyboard does different things depending on which mode the Notebook is in. There are two modes: edit mode and command mode.

Edit mode

Edit mode is indicated by a green cell border and a prompt showing in the editor area:



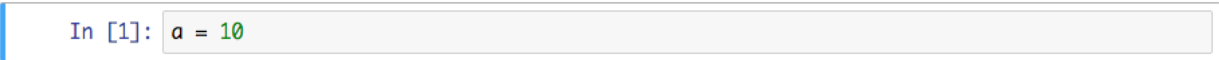
```
In [1]: a = 10
```

When a cell is in edit mode, you can type into the cell, like a normal text editor.

Enter edit mode by pressing **Enter** or using the mouse to click on a cell's editor area.

Command mode

Command mode is indicated by a grey cell border with a blue left margin:



```
In [1]: a = 10
```

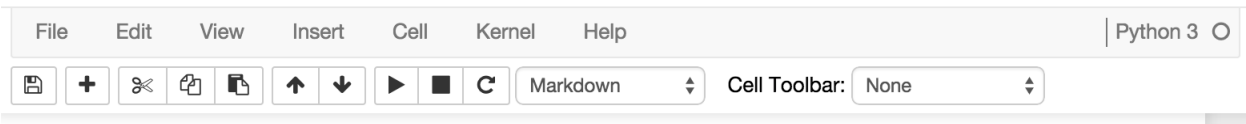
When you are in command mode, you are able to edit the notebook as a whole, but not type into individual cells. Most importantly, in command mode, the keyboard is mapped to a set of shortcuts that let you perform notebook and cell actions efficiently. For example, if you are in command mode and you press **c**, you will copy the current cell - no modifier is needed.

Don't try to type into a cell in command mode; unexpected things will happen!

Enter command mode by pressing **Esc** or using the mouse to click *outside* a cell's editor area.

Mouse navigation

All navigation and actions in the Notebook are available using the mouse through the menubar and toolbar, which are both above the main Notebook area:



The first idea of mouse based navigation is that **cells can be selected by clicking on them**. The currently selected cell gets a grey or green border depending on whether the notebook is in edit or command mode. If you click inside a cell's editor area, you will enter edit mode. If you click on the prompt or output area of a cell you will enter command mode.

If you are running this notebook in a live session (not on <http://nbviewer.jupyter.org>) try selecting different cells and going between edit and command mode. Try typing into a cell.

The second idea of mouse based navigation is that **cell actions usually apply to the currently selected cell**. Thus if you want to run the code in a cell, you would select it and click the

button in the toolbar or the “Cell:Run” menu item. Similarly, to copy a cell you would select it and click the

button in the toolbar or the “Edit:Copy” menu item. With this simple pattern, you should be able to do most everything you need with the mouse.

Markdown cells have one other state that can be modified with the mouse. These cells can either be rendered or unrendered. When they are rendered, you will see a nice formatted representation of the cell's contents. When they are unrendered, you will see the raw text source of the cell. To render the selected cell with the mouse, click the

button in the toolbar or the “Cell:Run” menu item. To unrender the selected cell, double click on the cell.

Keyboard Navigation

The modal user interface of the Jupyter Notebook has been optimized for efficient keyboard usage. This is made possible by having two different sets of keyboard shortcuts: one set that is active in edit mode and another in command mode.

The most important keyboard shortcuts are **Enter**, which enters edit mode, and **Esc**, which enters command mode.

In edit mode, most of the keyboard is dedicated to typing into the cell's editor. Thus, in edit mode there are relatively few shortcuts. In command mode, the entire keyboard is available for shortcuts, so there are many more. The `Help->Keyboard Shortcuts` dialog lists the available shortcuts.

We recommend learning the command mode shortcuts in the following rough order:

1. Basic navigation: `enter`, `shift-enter`, `up/k`, `down/j`
2. Saving the notebook: `s`
3. Change Cell types: `y`, `m`, `1-6`, `t`
4. Cell creation: `a`, `b`
5. Cell editing: `x`, `c`, `v`, `d`, `z`
6. Kernel operations: `i`, `0` (press twice)

1.4.3 Running Code

First and foremost, the Jupyter Notebook is an interactive environment for writing and running code. The notebook is capable of running code in a wide range of languages. However, each notebook is associated with a single kernel. This notebook is associated with the IPython kernel, therefore runs Python code.

Code cells allow you to enter and run code

Run a code cell using **Shift-Enter** or pressing the button in the toolbar above:

```
[2]: a = 10
```

```
[3]: print(a)
```

```
10
```

There are two other keyboard shortcuts for running code:

- **Alt-Enter** runs the current cell and inserts a new one below.
- **Ctrl-Enter** run the current cell and enters command mode.

Managing the Kernel

Code is run in a separate process called the Kernel. The Kernel can be interrupted or restarted. Try running the following cell and then hit the

button in the toolbar above.

```
[4]: import time
     time.sleep(10)
```

If the Kernel dies you will be prompted to restart it. Here we call the low-level system `libc.time` routine with the wrong argument via `ctypes` to segfault the Python interpreter:

```
[5]: import sys
     from ctypes import CDLL
     # This will crash a Linux or Mac system
     # equivalent calls can be made on Windows

     # Uncomment these lines if you would like to see the segfault

     # dll = 'dylib' if sys.platform == 'darwin' else 'so.6'
     # libc = CDLL("libc.%s" % dll)
     # libc.time(-1) # BOOM!!
```

Cell menu

The “Cell” menu has a number of menu items for running code in different ways. These includes:

- Run and Select Below
- Run and Insert Below
- Run All
- Run All Above
- Run All Below

Restarting the kernels

The kernel maintains the state of a notebook’s computations. You can reset this state by restarting the kernel. This is done by clicking on the

in the toolbar above.

sys.stdout and sys.stderr

The stdout and stderr streams are displayed as text in the output area.

```
[6]: print("hi, stdout")

hi, stdout
```

```
[7]: from __future__ import print_function
     print('hi, stderr', file=sys.stderr)
```

```
hi, stderr
```

Output is asynchronous

All output is displayed asynchronously as it is generated in the Kernel. If you execute the next cell, you will see the output one piece at a time, not all at the end.

```
[8]: import time, sys
      for i in range(8):
          print(i)
          time.sleep(0.5)
```

```
0
1
2
3
4
5
6
7
```

Large outputs

To better handle large outputs, the output area can be collapsed. Run the following cell and then single- or double-click on the active area to the left of the output:

```
[9]: for i in range(50):
      print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

(continues on next page)

(continued from previous page)

```
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
```

Beyond a certain point, output will scroll automatically:

```
[10]: for i in range(500):
      print(2**i - 1)
```

```
0
1
3
7
15
31
63
127
255
511
1023
2047
4095
8191
16383
32767
65535
131071
262143
524287
```

(continues on next page)

(continued from previous page)

```
1048575
2097151
4194303
8388607
16777215
33554431
67108863
134217727
268435455
536870911
1073741823
2147483647
4294967295
8589934591
17179869183
34359738367
68719476735
137438953471
274877906943
549755813887
1099511627775
2199023255551
4398046511103
8796093022207
17592186044415
35184372088831
70368744177663
140737488355327
281474976710655
562949953421311
1125899906842623
2251799813685247
4503599627370495
9007199254740991
18014398509481983
36028797018963967
72057594037927935
144115188075855871
288230376151711743
576460752303423487
1152921504606846975
2305843009213693951
4611686018427387903
9223372036854775807
18446744073709551615
36893488147419103231
73786976294838206463
147573952589676412927
295147905179352825855
590295810358705651711
1180591620717411303423
2361183241434822606847
```

(continues on next page)

(continued from previous page)

```

4722366482869645213695
9444732965739290427391
18889465931478580854783
37778931862957161709567
75557863725914323419135
151115727451828646838271
302231454903657293676543
604462909807314587353087
1208925819614629174706175
2417851639229258349412351
4835703278458516698824703
9671406556917033397649407
19342813113834066795298815
38685626227668133590597631
77371252455336267181195263
154742504910672534362390527
309485009821345068724781055
618970019642690137449562111
1237940039285380274899124223
2475880078570760549798248447
4951760157141521099596496895
9903520314283042199192993791
19807040628566084398385987583
39614081257132168796771975167
79228162514264337593543950335
158456325028528675187087900671
316912650057057350374175801343
633825300114114700748351602687
1267650600228229401496703205375
2535301200456458802993406410751
5070602400912917605986812821503
10141204801825835211973625643007
20282409603651670423947251286015
40564819207303340847894502572031
81129638414606681695789005144063
162259276829213363391578010288127
324518553658426726783156020576255
649037107316853453566312041152511
1298074214633706907132624082305023
2596148429267413814265248164610047
5192296858534827628530496329220095
10384593717069655257060992658440191
20769187434139310514121985316880383
41538374868278621028243970633760767
83076749736557242056487941267521535
166153499473114484112975882535043071
332306998946228968225951765070086143
664613997892457936451903530140172287
1329227995784915872903807060280344575
2658455991569831745807614120560689151
5316911983139663491615228241121378303
10633823966279326983230456482242756607

```

(continues on next page)

(continued from previous page)

```

21267647932558653966460912964485513215
42535295865117307932921825928971026431
85070591730234615865843651857942052863
170141183460469231731687303715884105727
340282366920938463463374607431768211455
680564733841876926926749214863536422911
1361129467683753853853498429727072845823
2722258935367507707706996859454145691647
5444517870735015415413993718908291383295
10889035741470030830827987437816582766591
21778071482940061661655974875633165533183
43556142965880123323311949751266331066367
87112285931760246646623899502532662132735
174224571863520493293247799005065324265471
348449143727040986586495598010130648530943
696898287454081973172991196020261297061887
1393796574908163946345982392040522594123775
2787593149816327892691964784081045188247551
5575186299632655785383929568162090376495103
11150372599265311570767859136324180752990207
22300745198530623141535718272648361505980415
44601490397061246283071436545296723011960831
89202980794122492566142873090593446023921663
178405961588244985132285746181186892047843327
356811923176489970264571492362373784095686655
713623846352979940529142984724747568191373311
1427247692705959881058285969449495136382746623
2854495385411919762116571938898990272765493247
570899077082383952423314387797980545530986495
11417981541647679048466287755595961091061972991
22835963083295358096932575511191922182123945983
45671926166590716193865151022383844364247891967
91343852333181432387730302044767688728495783935
182687704666362864775460604089535377456991567871
365375409332725729550921208179070754913983135743
730750818665451459101842416358141509827966271487
1461501637330902918203684832716283019655932542975
2923003274661805836407369665432566039311865085951
5846006549323611672814739330865132078623730171903
11692013098647223345629478661730264157247460343807
23384026197294446691258957323460528314494920687615
46768052394588893382517914646921056628989841375231
93536104789177786765035829293842113257979682750463
187072209578355573530071658587684226515959365500927
374144419156711147060143317175368453031918731001855
748288838313422294120286634350736906063837462003711
1496577676626844588240573268701473812127674924007423
2993155353253689176481146537402947624255349848014847
5986310706507378352962293074805895248510699696029695
11972621413014756705924586149611790497021399392059391
23945242826029513411849172299223580994042798784118783
47890485652059026823698344598447161988085597568237567

```

(continues on next page)

(continued from previous page)

```

95780971304118053647396689196894323976171195136475135
191561942608236107294793378393788647952342390272950271
383123885216472214589586756787577295904684780545900543
766247770432944429179173513575154591809369561091801087
153249554086588858358347027150309183618739122183602175
3064991081731777716716694054300618367237478244367204351
612998216346355543343388108601236734474956488734408703
12259964326927110866866776217202473468949912977468817407
24519928653854221733733552434404946937899825954937634815
49039857307708443467467104868809893875799651909875269631
98079714615416886934934209737619787751599303819750539263
196159429230833773869868419475239575503198607639501078527
392318858461667547739736838950479151006397215279002157055
784637716923335095479473677900958302012794430558004314111
1569275433846670190958947355801916604025588861116008628223
3138550867693340381917894711603833208051177722232017256447
6277101735386680763835789423207666416102355444464034512895
12554203470773361527671578846415332832204710888928069025791
25108406941546723055343157692830665664409421777856138051583
50216813883093446110686315385661331328818843555712276103167
100433627766186892221372630771322662657637687111424552206335
200867255532373784442745261542645325315275374222849104412671
401734511064747568885490523085290650630550748445698208825343
803469022129495137770981046170581301261101496891396417650687
1606938044258990275541962092341162602522202993782792835301375
3213876088517980551083924184682325205044405987565585670602751
6427752177035961102167848369364650410088811975131171341205503
12855504354071922204335696738729300820177623950262342682411007
25711008708143844408671393477458601640355247900524685364822015
51422017416287688817342786954917203280710495801049370729644031
102844034832575377634685573909834406561420991602098741459288063
205688069665150755269371147819668813122841983204197482918576127
411376139330301510538742295639337626245683966408394965837152255
822752278660603021077484591278675252491367932816789931674304511
1645504557321206042154969182557350504982735865633579863348609023
3291009114642412084309938365114701009965471731267159726697218047
6582018229284824168619876730229402019930943462534319453394436095
13164036458569648337239753460458804039861886925068638906788872191
26328072917139296674479506920917608079723773850137277813577744383
52656145834278593348959013841835216159447547700274555627155488767
105312291668557186697918027683670432318895095400549111254310977535
210624583337114373395836055367340864637790190801098222508621955071
421249166674228746791672110734681729275580381602196445017243910143
842498333348457493583344221469363458551160763204392890034487820287
1684996666696914987166688442938726917102321526408785780068975640575
3369993333393829974333376885877453834204643052817571560137951281151
6739986666787659948666753771754907668409286105635143120275902562303
13479973333575319897333507543509815336818572211270286240551805124607
26959946667150639794667015087019630673637144422540572481103610249215
53919893334301279589334030174039261347274288845081144962207220498431
107839786668602559178668060348078522694548577690162289924414440996863
215679573337205118357336120696157045389097155380324579848828881993727

```

(continues on next page)

(continued from previous page)

```

431359146674410236714672241392314090778194310760649159697657763987455
862718293348820473429344482784628181556388621521298319395315527974911
1725436586697640946858688965569256363112777243042596638790631055949823
3450873173395281893717377931138512726225554486085193277581262111899647
6901746346790563787434755862277025452451108972170386555162524223799295
13803492693581127574869511724554050904902217944340773110325048447598591
27606985387162255149739023449108101809804435888681546220650096895197183
55213970774324510299478046898216203619608871777363092441300193790394367
110427941548649020598956093796432407239217743554726184882600387580788735
220855883097298041197912187592864814478435487109452369765200775161577471
441711766194596082395824375185729628956870974218904739530401550323154943
883423532389192164791648750371459257913741948437809479060803100646309887
1766847064778384329583297500742918515827483896875618958121606201292619775
3533694129556768659166595001485837031654967793751237916243212402585239551
7067388259113537318333190002971674063309935587502475832486424805170479103
14134776518227074636666380005943348126619871175004951664972849610340958207
28269553036454149273332760011886696253239742350009903329945699220681916415
56539106072908298546665520023773392506479484700019806659891398441363832831
113078212145816597093331040047546785012958969400039613319782796882727665663
226156424291633194186662080095093570025917938800079226639565593765455331327
452312848583266388373324160190187140051835877600158453279131187530910662655
904625697166532776746648320380374280103671755200316906558262375061821325311
1809251394333065553493296640760748560207343510400633813116524750123642650623
3618502788666131106986593281521497120414687020801267626233049500247285301247
7237005577332262213973186563042994240829374041602535252466099000494570602495
14474011154664524427946373126085988481658748083205070504932198000989141204991
28948022309329048855892746252171976963317496166410141009864396001978282409983
57896044618658097711785492504343953926634992332820282019728792003956564819967
115792089237316195423570985008687907853269984665640564039457584007913129639935
231584178474632390847141970017375815706539969331281128078915168015826259279871
463168356949264781694283940034751631413079938662562256157830336031652518559743
926336713898529563388567880069503262826159877325124512315660672063305037119487
1852673427797059126777135760139006525652319754650249024631321344126610074238975
3705346855594118253554271520278013051304639509300498049262642688253220148477951
7410693711188236507108543040556026102609279018600996098525285376506440296955903
14821387422376473014217086081112052205218558037201992197050570753012880593911807
29642774844752946028434172162224104410437116074403984394101141506025761187823615
59285549689505892056868344324448208820874232148807968788202283012051522375647231
118571099379011784113736688648896417641748464297615937576404566024103044751294463
237142198758023568227473377297792835283496928595231875152809132048206089502588927
474284397516047136454946754595585670566993857190463750305618264096412179005177855
948568795032094272909893509191171341133987714380927500611236528192824358010355711
1897137590064188545819787018382342682267975428761855001222473056385648716020711423
3794275180128377091639574036764685364535950857523710002444946112771297432041422847
7588550360256754183279148073529370729071901715047420004889892225542594864082845695
15177100720513508366558296147058741458143803430094840009779784451085189728165691391
30354201441027016733116592294117482916287606860189680019559568902170379456331382783
60708402882054033466233184588234965832575213720379360039119137804340758912662765567
121416805764108066932466369176469931665150427440758720078238275608681517825325531135
242833611528216133864932738352939863330300854881517440156476551217363035650651062271
485667223056432267729865476705879726660601709763034880312953102434726071301302124543
971334446112864535459730953411759453321203419526069760625906204869452142602604249087

```

(continues on next page)

(continued from previous page)

```
1942668892225729070919461906823518906642406839052139521251812409738904285205208498175
3885337784451458141838923813647037813284813678104279042503624819477808570410416996351
7770675568902916283677847627294075626569627356208558085007249638955617140820833992703
15541351137805832567355695254588151253139254712417116170014499277911234281641667985407
3108270227561166513471139050917630250627850942483423234002899855582246856328335970815
62165404551223330269422781018352605012557018849668464680057997111644937126566671941631
124330809102446660538845562036705210025114037699336929360115994223289874253133343883263
248661618204893321077691124073410420050228075398673858720231988446579748506266687766527
497323236409786642155382248146820840100456150797347717440463976893159497012533375533055
994646472819573284310764496293641680200912301594695434880927953786318994025066751066111
1989292945639146568621528992587283360401824603189390869761855907572637988050133502132223
3978585891278293137243057985174566720803649206378781739523711815145275976100267004264447
7957171782556586274486115970349133441607298412757563479047423630290551952200534008528895
15914343565113172548972231940698266883214596825515126958094847260581103904401068017057791
31828687130226345097944463881396533766429193651030253916189694521162207808802136034115583
63657374260452690195888927762793067532858387302060507832379389042324415617604272068231167
127314748520905380391777855525586135065716774604121015664758778084648831235208544136462335
254629497041810760783555711051172270131433549208242031329517556169297662470417088272924671
509258994083621521567111422102344540262867098416484062659035112338595324940834176545849343
1018517988167243043134222844204689080525734196832968125318070224677190649881668353091698687
2037035976334486086268445688409378161051468393665936250636140449354381299763336706183397375
4074071952668972172536891376818756322102936787331872501272280898708762599526673412366794751
8148143905337944345073782753637512644205873574663745002544561797417525199053346824733589503
16296287810675888690147565507275025288411747149327490005089123594835050398106693649467179007
32592575621351777380295131014550050576823494298654980010178247189670100796213387298934358015
65185151242703554760590262029100101153646988597309960020356494379340201592426774597868716031
130370302485407109521180524058200202307293977194619920040712988758680403184853549195737432063
260740604970814219042361048116400404614587954389239840081425977517360806369707098391474864127
521481209941628438084722096232800809229175908778479680162851955034721612739414196782949728255
1042962419883256876169444192465601618458351817556959360325703910069443225478828393565899456511
2085924839766513752338888384931203236916703635113918720651407820138886450957656787131798913023
4171849679533027504677776769862406473833407270227837441302815640277772901915313574263597826047
8343699359066055009355553539724812947666814540455674882605631280555545803830627148527195652095
16687398718132110018711107079449625895333629080911349765211262561111091607661254297054391304191
3337479743626422003742221415889925179066725816182269953042252512222183215322508594108782608383
66749594872528440074844428317798503581334516323645399060845050244444366430645017188217565216767
133499189745056880149688856635597007162669032647290798121690100488888732861290034376435130433535
266998379490113760299377713271194014325338065294581596243380200977777465722580068752870260867071
533996758980227520598755426542388028650676130589163192486760401955554931445160137505740521734143
1067993517960455041197510853084776057301352261178326384973520803911109862890320275011481043468287
2135987035920910082395021706169552114602704522356652769947041607822219725780640550022962086936575
4271974071841820164790043412339104229205409044713305539894083215644439451561281100045924173873151
8543948143683640329580086824678208458410818089426611079788166431288878903122562200091848347746303
17087896287367280659160173649356416916821636178853222159576332862577757806245124400183696695492607
34175792574734561318320347298712833833643272357706444319152665725155515612490248800367393390985215
68351585149469122636640694597425667667286544715412888638305331450311031224980497600734786781970431
136703170298938245273281389194851335334573089430825777276610662900622062449960995201469573563940863
273406340597876490546562778389702670669146178861651554553221325801244124899921990402939147127881727
546812681195752981093125556779405341338292357723303109106442651602488249799843980805878294255763455
1093625362391505962186251113558810682676584715446606218212885303204976499599687961611756588511526911
2187250724783011924372502227117621365353169430893212436425770606409952999199375923223513177023053823
4374501449566023848745004454235242730706338861786424872851541212819905998398751846447026354046107647
```

(continues on next page)

(continued from previous page)

```

8749002899132047697490008908470485461412677723572849745703082425639811996797503692894052708092215295
17498005798264095394980017816940970922825355447145699491406164851279623993595007385788105416184430591
34996011596528190789960035633881941845650710894291398982812329702559247987190014771576210832368861183
69992023193056381579920071267763883691301421788582797965624659405118495974380029543152421664737722367
139984046386112763159840142535527767382602843577165595931249318810236991948760059086304843329475444735
27996809277225526319680285071055534765205687154331191862498637620473983897520118172609686658950889471
559936185544451052639360570142111069530411374308662383724997275240947967795040236345219373317901778943
1119872371088902105278721140284222139060822748617324767449994550481895935590080472690438746635803557887
2239744742177804210557442280568444278121645497234649534899989100963791871180160945380877493271607115775
4479489484355608421114884561136888556243290994469299069799978201927583742360321890761754986543214231551
8958978968711216842229769122273777112486581988938598139599956403855167484720643781523509973086428463103
1791795793742243368445953824454755422497316397787719627919991280771033496944128756304701994617285692620
3583591587484486736891907648909510844994632795575439255839982561542066993888257512609403989234571385241
7167183174968973473783815297819021689989265591150878511679965123084133987776515025218807978469142770483
143343663499379469475676305956380433799785311823017570233599302461682679755303005043761595693828554096
2866873269987589389513526119127608675995706236460351404671986049233653595110606010087523191387657108193
5733746539975178779027052238255217351991412472920702809343972098467307190221212020175046382775314216386
1146749307995035755805410447651043470398282494584140561868794419693461438044242404035009276555062843277
2293498615990071511610820895302086940796564989168281123737588839386922876088484808070018553110125686554
4586997231980143023221641790604173881593129978336562247475177678773845752176969616140037106220251373109
9173994463960286046443283581208347763186259956673124494950355357547691504353939232280074212440502746218
1834798892792057209288656716241669552637251991334624898990071071509538300870787846456014842488100549243
3669597785584114418577313432483339105274503982669249797980142143019076601741575692912029684976201098487
7339195571168228837154626864966678210549007965338499595960284286038153203483151385824059369952402196974
1467839114233645767430925372993335642109801593067699919192056857207630640696630277164811873990480439394
2935678228467291534861850745986671284219603186135399838384113714415261281393260554329623747980960878789
5871356456934583069723701491973342568439206372270799676768227428830522562786521108659247495961921757579
1174271291386916613944740298394668513687841274454159935353645485766104512557304221731849499192384351515
2348542582773833227889480596789337027375682548908319870707290971532209025114608443463698998384768703031
4697085165547666455778961193578674054751365097816639741414581943064418050229216886927397996769537406063
9394170331095332911557922387157348109502730195633279482829163886128836100458433773854795993539074812127
1878834066219066582311584477431469621900546039126655896565832777225767220091686754770959198707814962425
3757668132438133164623168954862939243801092078253311793131665554451534440183373509541918397415629924851
7515336264876266329246337909725878487602184156506623586263331108903068880366747019083836794831259849702
1503067252975253265849267581945175697520436831301324717252666221780613776073349403816767358966251969940
3006134505950506531698535163890351395040873662602649434505332443561227552146698807633534717932503939880
6012269011901013063397070327780702790081747325205298869010664887122455104293397615267069435865007879761
1202453802380202612679414065556140558016349465041059773802132977424491020858679523053413887173001575952
2404907604760405225358828131112281116032698930082119547604265954848982041717359046106827774346003151904
4809815209520810450717656262224562232065397860164239095208531909697964083434718092213655548692006303809
9619630419041620901435312524449124464130795720328478190417063819395928166869436184427311097384012607618
1923926083808324180287062504889824892826159144065695638083412763879185633373887236885462219476802521523
3847852167616648360574125009779649785652318288131391276166825527758371266747774473770924438953605043047
7695704335233296721148250019559299571304636576262782552333651055516742533495548947541848877907210086095
1539140867046659344229650003911859914260927315252556510466730211103348506699109789508369775581442017219
3078281734093318688459300007823719828521854630505113020933460422206697013398219579016739551162884034438
6156563468186637376918600015647439657043709261010226041866920844413394026796439158033479102325768068876
1231312693637327475383720003129487931408741852202045208373384168882678805359287831606695820465153613775
2462625387274654950767440006258975862817483704404090416746768337765357610718575663213391640930307227550
4925250774549309901534880012517951725634967408808180833493536675530715221437151326426783281860614455100
9850501549098619803069760025035903451269934817616361666987073351061430442874302652853566563721228910201
1970100309819723960613952005007180690253986963523272333397414670212286088574860530570713312744245782040

```

(continues on next page)

(continued from previous page)

```
39402006196394479212279040100143613805079739270465446667948293404245721771497210611414266254884915640800
7880401239278895842455808020028722761015947854093089333589658680849144354299442122282853250976983128161
1576080247855779168491161604005744552203189570818617866717931736169828870859888424456570650195396625632
3152160495711558336982323208011489104406379141637235733435863472339657741719776848913141300390793251264
6304320991423116673964646416022978208812758283274471466871726944679315483439553697826282600781586502529
126086419828462334792929283204595641762551656654894293374345388935863096687910739565256520156317300505
2521728396569246669585858566409191283525103313309788586748690777871726193375821479130513040312634601011
5043456793138493339171717132818382567050206626619577173497381555743452386751642958261026080625269202023
1008691358627698667834343426563676513410041325323915434699476311148690477350328591652205216125053840404
2017382717255397335668686853127353026820082650647830869398952622297380954700657183304410432250107680809
4034765434510794671337373706254706053640165301295661738797905244594761909401314366608820864500215361618
8069530869021589342674747412509412107280330602591323477595810489189523818802628733217641729000430723237
1613906173804317868534949482501882421456066120518264695519162097837904763760525746643528345800086144647
3227812347608635737069898965003764842912132241036529391038324195675809527521051493287056691600172289294
6455624695217271474139797930007529685824264482073058782076648391351619055042102986574113383200344578589
1291124939043454294827959586001505937164852896414611756415329678270323811008420597314822676640068915717
2582249878086908589655919172003011874329705792829223512830659356540647622016841194629645353280137831435
5164499756173817179311838344006023748659411585658447025661318713081295244033682389259290706560275662871
1032899951234763435862367668801204749731882317131689405132263742616259048806736477851858141312055132574
2065799902469526871724735337602409499463764634263378810264527485232518097613472955703716282624110265148
4131599804939053743449470675204818998927529268526757620529054970465036195226945911407432565248220530297
8263199609878107486898941350409637997855058537053515241058109940930072390453891822814865130496441060594
1652639921975621497379788270081927599571011707410703048211621988186014478090778364562973026099288212118
3305279843951242994759576540163855199142023414821406096423243976372028956181556729125946052198576424237
6610559687902485989519153080327710398284046829642812192846487952744057912363113458251892104397152848475
1322111937580497197903830616065542079656809365928562438569297590548811582472622691650378420879430569695
2644223875160994395807661232131084159313618731857124877138595181097623164945245383300756841758861139390
5288447750321988791615322464262168318627237463714249754277190362195246329890490766601513683517722278780
1057689550064397758323064492852433663725447492742849950855438072439049265978098153320302736703544455756
2115379100128795516646128985704867327450894985485699901710876144878098531956196306640605473407088911512
4230758200257591033292257971409734654901789970971399803421752289756197063912392613281210946814177823024
8461516400515182066584515942819469309803579941942799606843504579512394127824785226562421893628355646049
1692303280103036413316903188563893861960715988388559921368700915902478825564957045312484378725671129209
3384606560206072826633806377127787723921431976777119842737401831804957651129914090624968757451342258419
6769213120412145653267612754255575447842863953554239685474803663609915302259828181249937514902684516839
1353842624082429130653522550851115089568572790710847937094960732721983060451965636249987502980536903367
2707685248164858261307045101702230179137145581421695874189921465443966120903931272499975005961073806735
5415370496329716522614090203404460358274291162843391748379842930887932241807862544999950011922147613471
1083074099265943304522818040680892071654858232568678349675968586177586448361572508999990002384429522694
2166148198531886609045636081361784143309716465137356699351937172355172896723145017999980004768859045388
4332296397063773218091272162723568286619432930274713398703874344710345793446290035999960009537718090777
8664592794127546436182544325447136573238865860549426797407748689420691586892580071999920019075436181554
1732918558825509287236508865089427314647773172109885359481549737884138317378516014399984003815087236310
3465837117651018574473017730178854629295546344219770718963099475768276634757032028799968007630174472621
6931674235302037148946035460357709258591092688439541437926198951536553269514064057599936015260348945243
1386334847060407429789207092071541851718218537687908287585239790307310653902812811519987203052069789048
2772669694120814859578414184143083703436437075375816575170479580614621307805625623039974406104139578097
5545339388241629719156828368286167406872874150751633150340959161229242615611251246079948812208279156194
1109067877648325943831365673657233481374574830150326630068191832245848523122250249215989762441655831238
2218135755296651887662731347314466962749149660300653260136383664491697046244500498431979524883311662477
4436271510593303775325462694628933925498299320601306520272767328983394092489000996863959049766623324955
8872543021186607550650925389257867850996598641202613040545534657966788184978001993727918099533246649911
```

(continues on next page)

(continued from previous page)

```

1774508604237321510130185077851573570199319728240522608109106931593357636995600398745583619906649329982
3549017208474643020260370155703147140398639456481045216218213863186715273991200797491167239813298659964
7098034416949286040520740311406294280797278912962090432436427726373430547982401594982334479626597319929
1419606883389857208104148062281258856159455782592418086487285545274686109596480318996466895925319463985
2839213766779714416208296124562517712318911565184836172974571090549372219192960637992933791850638927971
5678427533559428832416592249125035424637823130369672345949142181098744438385921275985867583701277855943
1135685506711885766483318449825007084927564626073934469189828436219748887677184255197173516740255571188
2271371013423771532966636899650014169855129252147868938379656872439497775354368510394347033480511142377
4542742026847543065933273799300028339710258504295737876759313744878995550708737020788694066961022284754
9085484053695086131866547598600056679420517008591475753518627489757991101417474041577388133922044569509
1817096810739017226373309519720011335884103401718295150703725497951598220283494808315477626784408913901
3634193621478034452746619039440022671768206803436590301407450995903196440566989616630955253568817827803
7268387242956068905493238078880045343536413606873180602814901991806392881133979233261910507137635655607
1453677448591213781098647615776009068707282721374636120562980398361278576226795846652382101427527131121
2907354897182427562197295231552018137414565442749272241125960796722557152453591693304764202855054262243
5814709794364855124394590463104036274829130885498544482251921593445114304907183386609528405710108524486
1162941958872971024878918092620807254965826177099708896450384318689022860981436677321905681142021704897
2325883917745942049757836185241614509931652354199417792900768637378045721962873354643811362284043409794
4651767835491884099515672370483229019863304708398835585801537274756091443925746709287622724568086819588
9303535670983768199031344740966458039726609416797671171603074549512182887851493418575245449136173639177
186070713419675363980626894819329160794532188335934234320614909902436577570298683715049089827234727835
3721414268393507279612537896386583215890643766719068468641229819804873155140597367430098179654469455671
7442828536787014559225075792773166431781287533438136937282459639609746310281194734860196359308938911342
1488565707357402911845015158554633286356257506687627387456491927921949262056238946972039271861787782268
2977131414714805823690030317109266572712515013375254774912983855843898524112477893944078543723575564536
5954262829429611647380060634218533145425030026750509549825967711687797048224955787888157087447151129073
1190852565885922329476012126843706629085006005350101909965193542337559409644991157577631417489430225814
2381705131771844658952024253687413258170012010700203819930387084675118819289982315155262834978860451629
4763410263543689317904048507374826516340024021400407639860774169350237638579964630310525669957720903259
9526820527087378635808097014749653032680048042800815279721548338700475277159929260621051339915441806518
1905364105417475727161619402949930606536009608560163055944309667740095055431985852124210267983088361303
3810728210834951454323238805899861213072019217120326111888619335480190110863971704248420535966176722607
7621456421669902908646477611799722426144038434240652223777238670960380221727943408496841071932353445214
1524291284333980581729295522359944485228807686848130444755447734192076044345588681699368214386470689042
3048582568667961163458591044719888970457615373696260889510895468384152088691177363398736428772941378085
6097165137335922326917182089439777940915230747392521779021790936768304177382354726797472857545882756171
1219433027467184465383436417887955588183046149478504355804358187353660835476470945359494571509176551234
2438866054934368930766872835775911176366092298957008711608716374707321670952941890718989143018353102468
4877732109868737861533745671551822352732184597914017423217432749414643341905883781437978286036706204937
9755464219737475723067491343103644705464369195828034846434865498829286683811767562875956572073412409874
1951092843947495144613498268620728941092873839165606969286973099765857336762353512575191314414682481974
3902185687894990289226996537241457882185747678331213938573946199531714673524707025150382628829364963949
7804371375789980578453993074482915764371495356662427877147892399063429347049414050300765257658729927899
1560874275157996115690798614896583152874299071332485575429578479812685869409882810060153051531745985579
3121748550315992231381597229793166305748598142664971150859156959625371738819765620120306103063491971159
6243497100631984462763194459586332611497196285329942301718313919250743477639531240240612206126983942319
1248699420126396892552638891917266522299439257065988460343662783850148695527906248048122441225396788463
2497398840252793785105277783834533044598878514131976920687325567700297391055812496096244882450793576927
4994797680505587570210555567669066089197757028263953841374651135400594782111624992192489764901587153855
998959536101117514042111135338132178395514056527907682749302270801189564223249984384979529803174307711
199791907220223502808422227067626435679102811305581536549860454160237912844649996876995905960634861542
3995838144404470056168444454135252871358205622611163073099720908320475825689299993753991811921269723084

```

(continues on next page)

(continued from previous page)

```

7991676288808940112336888908270505742716411245222326146199441816640951651378599987507983623842539446169
1598335257761788022467377781654101148543282249044465229239888363328190330275719997501596724768507889233
3196670515523576044934755563308202297086564498088930458479776726656380660551439995003193449537015778467
6393341031047152089869511126616404594173128996177860916959553453312761321102879990006386899074031556935
1278668206209430417973902225323280918834625799235572183391910690662552264220575998001277379814806311387
2557336412418860835947804450646561837669251598471144366783821381325104528441151996002554759629612622774
5114672824837721671895608901293123675338503196942288733567642762650209056882303992005109519259225245548
1022934564967544334379121780258624735067700639388457746713528552530041811376460798401021903851845049109
2045869129935088668758243560517249470135401278776915493427057105060083622752921596802043807703690098219
4091738259870177337516487121034498940270802557553830986854114210120167245505843193604087615407380196438
8183476519740354675032974242068997880541605115107661973708228420240334491011686387208175230814760392877
1636695303948070935006594848413799576108321023021532394741645684048066898202337277441635046162952078575

```

1.4.4 Markdown Cells

Text can be added to Jupyter Notebooks using Markdown cells. You can change the cell type to Markdown by using the Cell menu, the toolbar, or the key shortcut `m`. Markdown is a popular markup language that is a superset of HTML. Its specification can be found here:

<https://daringfireball.net/projects/markdown/>

Markdown basics

You can make text *italic* or **bold** by surrounding a block of text with a single or double `*` respectively

You can build nested itemized or enumerated lists:

- One
 - Sublist
 - * This
- Sublist - That - The other thing
- Two
- Sublist
- Three
- Sublist

Now another list:

1. Here we go
 1. Sublist
 2. Sublist
2. There we go
3. Now this

You can add horizontal rules:

Here is a blockquote:

Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one— and preferably only one —obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than *right* now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea – let's do more of those!

And shorthand for links:

[Jupyter's website](#)

You can use backslash to generate literal characters which would otherwise have special meaning in the Markdown syntax.

```
\*literal asterisks\*  
*literal asterisks*
```

Use double backslash to generate the literal \$ symbol.

Headings

You can add headings by starting a line with one (or multiple) # followed by a space, as in the following example:

```
# Heading 1  
# Heading 2  
## Heading 2.1  
## Heading 2.2
```

Embedded code

You can embed code meant for illustration instead of execution in Python:

```
def f(x):  
    """a docstring"""  
    return x**2
```

or other languages:

```
for (i=0; i<n; i++) {  
    printf("hello %d\n", i);  
    x += 4;  
}
```

LaTeX equations

Courtesy of MathJax, you can include mathematical expressions both inline: $e^{i\pi} + 1 = 0$ and displayed:

$$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i \quad (1.1)$$

Inline expressions can be added by surrounding the latex code with \$:

```
$e^{i\pi} + 1 = 0$
```

Expressions on their own line are surrounded by `\begin{equation}` and `\end{equation}`:

```
\begin{equation}
e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i
\end{equation}
```

GitHub flavored markdown

The Notebook webapp supports Github flavored markdown meaning that you can use triple backticks for code blocks:

```
```python
print "Hello World"
```

```javascript
console.log("Hello World")
```
```

Gives:

```
print "Hello World"
```

```
console.log("Hello World")
```

And a table like this:

```
This	is
a	table
```

A nice HTML Table:

| | |
|------|-------|
| This | is |
| a | table |

General HTML

Because Markdown is a superset of HTML you can even add things like HTML tables:

Header 1

Header 2

row 1, cell 1

row 1, cell 2

row 2, cell 1

row 2, cell 2

Local files

If you have local files in your Notebook directory, you can refer to these files in Markdown cells directly:

```
[subdirectory/]<filename>
```

For example, in the images folder, we have the Python logo:

```

```

and a video with the HTML5 video tag:

```
<video controls src="../../../images/animation.m4v">animation</video>
```

animation

These do not embed the data into the notebook file, and require that the files exist when you are viewing the notebook.

Security of local files

Note that this means that the Jupyter notebook server also acts as a generic file server for files inside the same tree as your notebooks. Access is not granted outside the notebook folder so you have strict control over what files are visible, but for this reason it is highly recommended that you do not run the notebook server with a notebook directory at a high level in your filesystem (e.g. your home directory).

When you run the notebook in a password-protected manner, local file access is restricted to authenticated users unless read-only views are active.

Markdown attachments

Since Jupyter notebook version 5.0, in addition to referencing external file you can attach a file to a markdown cell. To do so drag the file from in a markdown cell while editing it:



Files are stored in cell metadata and will be automatically scrubbed at save-time if not referenced. You can recognize attached images from other files by their url that starts with `attachment:`. For the image above:

```
![pycon-logo.jpg](attachment:pycon-logo.jpg)
```

Keep in mind that attached files will increase the size of your notebook.

You can manually edit the attachment by using the `View > Cell Toolbar > Attachment` menu, but you should not need to.

1.4.5 Keyboard Shortcut Customization

Starting with Jupyter Notebook 5.0, you can customize the `command` mode shortcuts from within the Notebook Application itself.

Head to the `Help` menu and select the `Edit keyboard Shortcuts` item. A dialog will guide you through the process of adding custom keyboard shortcuts.

Keyboard shortcut set from within the Notebook Application will be persisted to your configuration file. A single action may have several shortcuts attached to it.

1.4.6 Keyboard Shortcut Customization (Pre Notebook 5.0)

Starting with IPython 2.0 keyboard shortcuts in command and edit mode are fully customizable. These customizations are made using the Jupyter JavaScript API. Here is an example that makes the `r` key available for running a cell:

```
[ ]: %%javascript

Jupyter.keyboard_manager.command_shortcuts.add_shortcut('r', {
  help : 'run cell',
  help_index : 'zz',
  handler : function (event) {
    IPython.notebook.execute_cell();
    return false;
  }}
);
```

“By default the keypress `r`, while in command mode, changes the type of the selected cell to `raw`. This shortcut is overridden by the code in the previous cell, and thus the action no longer be available via the keypress `r`.”

There are a couple of points to mention about this API:

- The `help_index` field is used to sort the shortcuts in the Keyboard Shortcuts help dialog. It defaults to `zz`.
- When a handler returns `false` it indicates that the event should stop propagating and the default action should not be performed. For further details about the `event` object or event handling, see the jQuery docs.
- If you don't need a `help` or `help_index` field, you can simply pass a function as the second argument to `add_shortcut`.

```
[ ]: %%javascript

Jupyter.keyboard_manager.command_shortcuts.add_shortcut('r', function (event) {
  IPython.notebook.execute_cell();
  return false;
});
```

Likewise, to remove a shortcut, use `remove_shortcut`:

```
[ ]: %%javascript

Jupyter.keyboard_manager.command_shortcuts.remove_shortcut('r');
```

If you want your keyboard shortcuts to be active for all of your notebooks, put the above API calls into your `custom.js` file.

Of course we provide name for majority of existing action so that you do not have to re-write everything, here is for example how to bind `r` back to it's initial behavior:

```
[ ]: %%javascript

Jupyter.keyboard_manager.command_shortcuts.add_shortcut('r', 'jupyter-notebook:change-
↩cell-to-raw');
```

1.4.7 Embracing web standards

One of the main reasons why we developed the current notebook web application was to embrace the web technology.

By being a pure web application using HTML, JavaScript, and CSS, the Notebook can get all the web technology improvement for free. Thus, as browser support for different media extend, the notebook web app should be able to be compatible without modification.

This is also true with performance of the User Interface as the speed of JavaScript VM increases.

The other advantage of using only web technology is that the code of the interface is fully accessible to the end user and is modifiable live. Even if this task is not always easy, we strive to keep our code as accessible and reusable as possible. This should allow us - with minimum effort - development of small extensions that customize the behavior of the web interface.

Tampering with the Notebook application

The first tool that is available to you and that you should be aware of are browser “developers tool”. The exact naming can change across browser and might require the installation of extensions. But basically they can allow you to inspect/modify the DOM, and interact with the JavaScript code that runs the frontend.

- In Chrome and Safari, Developer tools are in the menu **View > Developer > JavaScript Console**
- In Firefox you might need to install [Firebug](#)

Those will be your best friends to debug and try different approaches for your extensions.

Injecting JS

Using magics

The above tools can be tedious for editing edit long JavaScript files. Therefore we provide the `%%javascript` magic. This allows you to quickly inject JavaScript into the notebook. Still the JavaScript injected this way will not survive reloading. Hence, it is a good tool for testing and refining a script.

You might see here and there people modifying css and injecting js into the notebook by reading file(s) and publishing them into the notebook. Not only does this often break the flow of the notebook and make the re-execution of the notebook broken, but it also means that you need to execute those cells in the entire notebook every time you need to update the code.

This can still be useful in some cases, like the `%autosave` magic that allows you to control the time between each save. But this can be replaced by a JavaScript dropdown menu to select the save interval.

```
[ ]: ## you can inspect the autosave code to see what it does.
      %autosave??
```

custom.js

To inject JavaScript we provide an entry point: `custom.js` that allows the user to execute and load other resources into the notebook. JavaScript code in `custom.js` will be executed when the notebook app starts and can then be used to customize almost anything in the UI and in the behavior of the notebook.

`custom.js` can be found in the `~/.jupyter/custom/custom.js`. You can share your `custom.js` with others.

Back to theory

```
[ ]: from jupyter_core.paths import jupyter_config_dir
jupyter_dir = jupyter_config_dir()
jupyter_dir
```

and `custom.js` is in

```
[ ]: import os.path
custom_js_path = os.path.join(jupyter_dir, 'custom', 'custom.js')
```

```
[ ]: # my custom js
if os.path.isfile(custom_js_path):
    with open(custom_js_path) as f:
        print(f.read())
else:
    print("You don't have a custom.js file")
```

Note that `custom.js` is meant to be modified by user. When writing a script, you can define it in a separate file and add a line of configuration into `custom.js` that will fetch and execute the file.

Warning : even if modification of `custom.js` takes effect immediately after browser refresh (except if browser cache is aggressive), *creating* a file in `static/` directory needs a **server restart**.

Exercise :

- Create a `custom.js` in the right location with the following content:

```
alert("hello world from custom.js")
```

- Restart your server and open any notebook.
- Be greeted by `custom.js`

Have a look at `default custom.js`, to see it's content and for more explanation.

For the quick ones :

We've seen above that you can change the autosave rate by using a magic. This is typically something I don't want to type every time, and that I don't like to embed into my workflow and documents. (readers don't care what my autosave time is). Let's build an extension that allows us to do it.

Create a dropdown element in the toolbar (DOM `Jupyter.toolbar.element`), you will need

- `Jupyter.notebook.set_autosave_interval(milliseconds)`
- know that 1 min = 60 sec, and 1 sec = 1000 ms

```

var label = jQuery('<label/>').text('AutoScroll Limit:');
var select = jQuery('<select/>')
    // .append(jQuery('<option/>').attr('value', '2').text('2min (default)'))
    .append(jQuery('<option/>').attr('value', undefined).text('disabled'))

    // TODO:
    // the_toolbar_element.append(label)
    // the_toolbar_element.append(select);

select.change(function() {
    var val = jQuery(this).val() // val will be the value in [2]
    // TODO
    // this will be called when dropdown changes
});

var time_m = [1,5,10,15,30];
for (var i=0; i < time_m.length; i++) {
    var ts = time_m[i];

    // [2]    ____ this will be `val` on [1]
    //      |
    //      v

    select.append($('<option/>').attr('value', ts).text(thr+'min'));
    // this will fill up the dropdown `select` with
    // 1 min
    // 5 min
    // 10 min
    // 10 min
    // ...
}

```

A non-interactive example first

I like my cython to be nicely highlighted

```

Jupyter.config.cell_magic_highlight['magic_text/x-cython'] = {}
Jupyter.config.cell_magic_highlight['magic_text/x-cython'].reg = [/^%%cython/]

```

`text/x-cython` is the name of CodeMirror mode name, `magic_` prefix will just patch the mode so that the first line that contains a magic does not screw up the highlighting. `reg` is a list or regular expression that will trigger the change of mode.

Get more documentation

Sadly, you will have to read the js source file (but there are lots of comments) and/or build the JavaScript documentation using yuidoc. If you have node and yui-doc installed:

```
$ cd ~/jupyter/nbclassic/nbclassic/static/notebook/js/
$ yuidoc . --server
warn: (yuidoc): Failed to extract port, setting to the default :3000
info: (yuidoc): Starting YUIDoc@0.3.45 using YUI@3.9.1 with NodeJS@0.10.15
info: (yuidoc): Scanning for yuidoc.json file.
info: (yuidoc): Starting YUIDoc with the following options:
info: (yuidoc):
{ port: 3000,
  nocode: false,
  paths: [ '.' ],
  server: true,
  outdir: './out' }
info: (yuidoc): Scanning for yuidoc.json file.
info: (server): Starting server: http://127.0.0.1:3000
```

and browse <http://127.0.0.1:3000> to get documentation

Some convenience methods

By browsing the documentation you will see that we have some convenience methods that allows us to avoid re-inventing the UI every time :

```
Jupyter.toolbar.add_buttons_group([
    {
        'label'    : 'run qtconsole',
        'icon'     : 'fa-terminal', // select your icon from
                                   // http://fontawesome.io/icons/
        'callback': function(){Jupyter.notebook.kernel.execute('%qtconsole')}
    }
    // add more button here if needed.
]);
```

with a lot of icons you can select from.

Cell Metadata

The most requested feature is generally to be able to distinguish an individual cell in the notebook, or run a specific action with them. To do so, you can either use `Jupyter.notebook.get_selected_cell()`, or rely on `CellToolbar`. This allows you to register a set of actions and graphical elements that will be attached to individual cells.

Cell Toolbar

You can see some example of what can be done by toggling the Cell Toolbar selector in the toolbar on top of the notebook. It provides two default presets that are Default and slideshow. Default allows the user to edit the metadata attached to each cell manually.

First we define a function that takes at first parameter an element on the DOM in which to inject UI element. The second element is the cell this element was registered with. Then we will need to register that function and give it a name.

Register a callback

```
[ ]: %%javascript
var CellToolbar = Jupyter.CellToolbar
var toggle = function(div, cell) {
    var button_container = $(div)

    // let's create a button that shows the current value of the metadata
    var button = $('<button/>').addClass('btn btn-mini').text(String(cell.metadata.
↵foo));

    // On click, change the metadata value and update the button label
    button.click(function(){
        var v = cell.metadata.foo;
        cell.metadata.foo = !v;
        button.text(String(!v));
    })

    // add the button to the DOM div.
    button_container.append(button);
}

// now we register the callback under the name foo to give the
// user the ability to use it later
CellToolbar.register_callback('tuto.foo', toggle);
```

Registering a preset

This function can now be part of many preset of the CellToolBar.

```
[ ]: %%javascript
Jupyter.CellToolbar.register_preset('Tutorial 1',['tuto.foo','default.rawedit'])
Jupyter.CellToolbar.register_preset('Tutorial 2',['slideshow.select','tuto.foo'])
```

You should now have access to two presets :

- Tutorial 1
- Tutorial 2

And check that the buttons you defined share state when you toggle preset. Also check that the metadata of the cell is modified when you click the button, and that when saved on reloaded the metadata is still available.

Exercise:

Try to wrap the all code in a file, put this file in `{jupyter_dir}/custom/<a-name>.js`, and add

```
require(['custom/<a-name>']);
```

in `custom.js` to have this script automatically loaded in all your notebooks.

`require` is provided by a [JavaScript library](#) that allow you to express dependency. For simple extension like the previous one we directly mute the global namespace, but for more complex extension you could pass a callback to `require([...], <callback>)` call, to allow the user to pass configuration information to your plugin.

In Python language,

```
require(['a/b', 'c/d'], function( e, f){
    e.something()
    f.something()
})
```

could be read as

```
import a.b as e
import c.d as f
e.something()
f.something()
```

See for example @damianavila “ZenMode” plugin :

```
// read that as
// import custom.zenmode.main as zenmode
require(['custom/zenmode/main'], function(zenmode){
    zenmode.background('images/back12.jpg');
})
```

For the quickest

Try to use [the following](#) to bind a dropdown list to `cell.metadata.difficulty.select`.

It should be able to take the 4 following values :

- <None>
- Easy
- Medium
- Hard

We will use it to customize the output of the converted notebook depending on the tag on each cell

```
[1]: # %load soln/celldiff.js
```

```
[ ]:
```


1.4.8 Importing Jupyter Notebooks as Modules

It is a common problem that people want to import code from Jupyter Notebooks. This is made difficult by the fact that Notebooks are not plain Python files, and thus cannot be imported by the regular Python machinery.

Fortunately, Python provides some fairly sophisticated [hooks](#) into the import machinery, so we can actually make Jupyter notebooks importable without much difficulty, and only using public APIs.

```
[ ]: import io, os, sys, types
```

```
[ ]: from IPython import get_ipython
     from nbformat import read
     from IPython.core.interactiveshell import InteractiveShell
```

Import hooks typically take the form of two objects:

1. a Module **Loader**, which takes a module name (e.g. 'IPython.display'), and returns a Module
2. a Module **Finder**, which figures out whether a module might exist, and tells Python what **Loader** to use

```
[ ]: def find_notebook(fullname, path=None):
    """find a notebook, given its fully qualified name and an optional path

    This turns "foo.bar" into "foo/bar.ipynb"
    and tries turning "Foo_Bar" into "Foo Bar" if Foo_Bar
    does not exist.
    """
    name = fullname.rsplit('.', 1)[-1]
    if not path:
        path = ['']
    for d in path:
        nb_path = os.path.join(d, name + ".ipynb")
        if os.path.isfile(nb_path):
            return nb_path
        # let import Notebook_Name find "Notebook Name.ipynb"
        nb_path = nb_path.replace("-", " ")
        if os.path.isfile(nb_path):
            return nb_path
```

Notebook Loader

Here we have our Notebook Loader. It's actually quite simple - once we figure out the filename of the module, all it does is:

1. load the notebook document into memory
2. create an empty Module
3. execute every cell in the Module namespace

Since IPython cells can have extended syntax, the IPython transform is applied to turn each of these cells into their pure-Python counterparts before executing them. If all of your notebook cells are pure-Python, this step is unnecessary.

```
[ ]: class NotebookLoader(object):
    """Module Loader for Jupyter Notebooks"""
    def __init__(self, path=None):
```

(continues on next page)

(continued from previous page)

```

self.shell = InteractiveShell.instance()
self.path = path

def load_module(self, fullname):
    """import a notebook as a module"""
    path = find_notebook(fullname, self.path)

    print ("importing Jupyter notebook from %s" % path)

    # load the notebook object
    with io.open(path, 'r', encoding='utf-8') as f:
        nb = read(f, 4)

    # create the module and add it to sys.modules
    # if name in sys.modules:
    #     return sys.modules[name]
    mod = types.ModuleType(fullname)
    mod.__file__ = path
    mod.__loader__ = self
    mod.__dict__['get_ipython'] = get_ipython
    sys.modules[fullname] = mod

    # extra work to ensure that magics that would affect the user_ns
    # actually affect the notebook module's ns
    save_user_ns = self.shell.user_ns
    self.shell.user_ns = mod.__dict__

    try:
        for cell in nb.cells:
            if cell.cell_type == 'code':
                # transform the input to executable Python
                code = self.shell.input_transformer_manager.transform_cell(cell.source)
                # run the code in the module
                exec(code, mod.__dict__)
    finally:
        self.shell.user_ns = save_user_ns
    return mod

```

The Module Finder

The finder is a simple object that tells you whether a name can be imported, and returns the appropriate loader. All this one does is check, when you do:

```
import mynotebook
```

it checks whether `mynotebook.ipynb` exists. If a notebook is found, then it returns a `NotebookLoader`.

Any extra logic is just for resolving paths within packages.

```
[ ]: class NotebookFinder(object):
    """Module finder that locates Jupyter Notebooks"""
    def __init__(self):
        self.loaders = {}

    def find_module(self, fullname, path=None):
        nb_path = find_notebook(fullname, path)
        if not nb_path:
            return

        key = path
        if path:
            # lists aren't hashable
            key = os.path.sep.join(path)

        if key not in self.loaders:
            self.loaders[key] = NotebookLoader(path)
        return self.loaders[key]
```

Register the hook

Now we register the NotebookFinder with `sys.meta_path`

```
[ ]: sys.meta_path.append(NotebookFinder())
```

After this point, my notebooks should be importable.

Let's look at what we have in the CWD:

```
[ ]: ls nbpackage
```

So I should be able to import `nbpackage.mynotebook`.

```
[ ]: import nbpackage.mynotebook
```

Aside: displaying notebooks

Here is some simple code to display the contents of a notebook with syntax highlighting, etc.

```
[ ]: from pygments import highlight
    from pygments.lexers import PythonLexer
    from pygments.formatters import HtmlFormatter

    from IPython.display import display, HTML

    formatter = HtmlFormatter()
    lexer = PythonLexer()

    # publish the CSS for pygments highlighting
    display(HTML("""
```

(continues on next page)

(continued from previous page)

```
<style type='text/css'>
%s
</style>
""" % formatter.get_style_defs()
))
```

```
[ ]: def show_notebook(fname):
    """display a short summary of the cells of a notebook"""
    with io.open(fname, 'r', encoding='utf-8') as f:
        nb = read(f, 4)
        html = []
        for cell in nb.cells:
            html.append("<h4>%s cell</h4>" % cell.cell_type)
            if cell.cell_type == 'code':
                html.append(highlight(cell.source, lexer, formatter))
            else:
                html.append("<pre>%s</pre>" % cell.source)
        display(HTML('\n'.join(html)))

show_notebook(os.path.join("nbpackage", "mynotebook.ipynb"))
```

So my notebook has some code cells, one of which contains some IPython syntax.

Let's see what happens when we import it

```
[ ]: from nbpackage import mynotebook
```

Hooray, it imported! Does it work?

```
[ ]: mynotebook.foo()
```

Hooray again!

Even the function that contains IPython syntax works:

```
[ ]: mynotebook.has_ip_syntax()
```

Notebooks in packages

We also have a notebook inside the nb package, so let's make sure that works as well.

```
[ ]: ls nbpackage/nbs
```

Note that the `__init__.py` is necessary for nb to be considered a package, just like usual.

```
[ ]: show_notebook(os.path.join("nbpackage", "nbs", "other.ipynb"))
```

```
[ ]: from nbpackage.nbs import other
other.bar(5)
```

So now we have importable notebooks, from both the local directory and inside packages.

I can even put a notebook inside IPython, to further demonstrate that this is working properly:

```
[ ]: import shutil
from IPython.paths import get_ipython_package_dir

utils = os.path.join(get_ipython_package_dir(), 'utils')
shutil.copy(os.path.join("nbpackage", "mynotebook.ipynb"),
            os.path.join(utils, "inside_ipython.ipynb")
)
```

and import the notebook from IPython.utils

```
[ ]: from IPython.utils import inside_ipython
inside_ipython.whatsmyname()
```

This approach can even import functions and classes that are defined in a notebook using the %%cython magic.

1.4.9 Connecting to an existing IPython kernel using the Qt Console

The Frontend/Kernel Model

The traditional IPython (ipython) consists of a single process that combines a terminal based UI with the process that runs the users code.

While this traditional application still exists, the modern Jupyter consists of two processes:

- Kernel: this is the process that runs the users code.
- Frontend: this is the process that provides the user interface where the user types code and sees results.

Jupyter currently has 3 frontends:

- Terminal Console (jupyter console)
- Qt Console (jupyter qtconsole)
- Notebook (jupyter notebook)

The Kernel and Frontend communicate over a ZeroMQ/JSON based messaging protocol, which allows multiple Frontends (even of different types) to communicate with a single Kernel. This opens the door for all sorts of interesting things, such as connecting a Console or Qt Console to a Notebook's Kernel. For example, you may want to connect a Qt console to your Notebook's Kernel and use it as a help browser, calling ?? on objects in the Qt console (whose pager is more flexible than the one in the notebook).

This Notebook describes how you would connect another Frontend to an IPython Kernel that is associated with a Notebook. The commands currently given here are specific to the IPython kernel.

Manual connection

To connect another Frontend to a Kernel manually, you first need to find out the connection information for the Kernel using the %connect_info magic:

```
[ ]: %connect_info
```

You can see that this magic displays everything you need to connect to this Notebook's Kernel.

Automatic connection using a new Qt Console

You can also start a new Qt Console connected to your current Kernel by using the `%qtconsole` magic. This will detect the necessary connection information and start the Qt Console for you automatically.

```
[ ]: a = 10
```

```
[ ]: %qtconsole
```

The Markdown parser included in the Jupyter Notebook is MathJax-aware. This means that you can freely mix in mathematical expressions using the [MathJax subset of Tex and LaTeX](#). Some examples from the [MathJax demos site](#) are reproduced below, as well as the Markdown+TeX source.

1.4.10 Motivating Examples

The Lorenz Equations

Source

```
\begin{align}
\dot{x} &= \sigma(y-x) \\
\dot{y} &= \rho x - y - xz \\
\dot{z} &= -\beta z + xy
\end{align}
```

Display

The Cauchy-Schwarz Inequality

Source

```
\begin{equation*}
\left( \sum_{k=1}^n a_k b_k \right)^2 \leq \left( \sum_{k=1}^n a_k^2 \right) \left( \sum_{k=1}^n b_k^2 \right)
\end{equation*}
```

Display

$$\left(\sum_{k=1}^n a_k b_k \right)^2 \leq \left(\sum_{k=1}^n a_k^2 \right) \left(\sum_{k=1}^n b_k^2 \right)$$

A Cross Product Formula

Source

```
\begin{equation*}
\mathbf{V}_1 \times \mathbf{V}_2 = \begin{vmatrix}
\mathbf{i} & \mathbf{j} & \mathbf{k} \\
\frac{\partial X}{\partial u} & \frac{\partial Y}{\partial u} & 0 \\
\frac{\partial X}{\partial v} & \frac{\partial Y}{\partial v} & 0
\end{vmatrix}
\end{equation*}
```

Display

$$\mathbf{V}_1 \times \mathbf{V}_2 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \frac{\partial X}{\partial u} & \frac{\partial Y}{\partial u} & 0 \\ \frac{\partial X}{\partial v} & \frac{\partial Y}{\partial v} & 0 \end{vmatrix}$$

The probability of getting (k) heads when flipping (n) coins is

Source

```
\begin{equation*}
P(E) = \{n \choose k\} p^k (1-p)^{n-k}
\end{equation*}
```

Display

$$P(E) = \binom{n}{k} p^k (1-p)^{n-k}$$

An Identity of Ramanujan

Source

```
\begin{equation*}
\frac{1}{\operatorname{Bigl}(\sqrt{\phi \sqrt{5}}-\phi)\operatorname{Bigr)} e^{\frac{25}{\pi}}} =
1+\frac{e^{-2\pi}}{1+\frac{e^{-4\pi}}{1+\frac{e^{-6\pi}}{1+\frac{e^{-8\pi}}{1+\ldots}}}}}
\end{equation*}
```

Display

$$\frac{1}{\left(\sqrt{\phi\sqrt{5}-\phi}\right)e^{\frac{2}{3}\pi}} = 1 + \frac{e^{-2\pi}}{1 + \frac{e^{-4\pi}}{1 + \frac{e^{-6\pi}}{1 + \frac{e^{-8\pi}}{1 + \dots}}}}$$

A Rogers-Ramanujan Identity

Source

```
\begin{equation*}
1 + \frac{q^2}{(1-q)} + \frac{q^6}{(1-q)(1-q^2)} + \cdots =
\prod_{j=0}^{\infty} \frac{1}{(1-q^{5j+2})(1-q^{5j+3})},
\quad \text{for } |q| < 1.
\end{equation*}
```

Display

$$1 + \frac{q^2}{(1-q)} + \frac{q^6}{(1-q)(1-q^2)} + \cdots = \prod_{j=0}^{\infty} \frac{1}{(1-q^{5j+2})(1-q^{5j+3})}, \quad \text{for } |q| < 1.$$

Maxwell's Equations

Source

```
\begin{align}
\nabla \times \vec{\mathbf{B}} &= \frac{1}{c} \frac{\partial \vec{\mathbf{E}}}{\partial t} \\
\nabla \cdot \vec{\mathbf{E}} &= \frac{4\pi}{c} \rho \\
\nabla \times \vec{\mathbf{E}} &= -\frac{1}{c} \frac{\partial \vec{\mathbf{B}}}{\partial t} \\
\nabla \cdot \vec{\mathbf{B}} &= 0
\end{align}
```

Display

Equation Numbering and References

Equation numbering and referencing will be available in a future version of the Jupyter notebook.

Inline Typesetting (Mixing Markdown and TeX)

While display equations look good for a page of samples, the ability to mix math and *formatted text* in a paragraph is also important.

Source

This expression $\sqrt{3x-1}+(1+x)^2$ is an example of a TeX inline equation in a [\[Markdown-formatted\]](https://daringfireball.net/projects/markdown/) (https://daringfireball.net/projects/markdown/) sentence.

Display

This expression $\sqrt{3x-1}+(1+x)^2$ is an example of a TeX inline equation in a [Markdown-formatted](#) sentence.

Other Syntax

You will notice in other places on the web that $\$$ are needed explicitly to begin and end MathJax typesetting. This is **not** required if you will be using TeX environments, but the Jupyter notebook will accept this syntax on legacy notebooks.

Source

```
$$
\begin{array}{c}
y_1 \\\
y_2 \mathtt{t}_i \\\
z_{3,4}
\end{array}
$$
```

```
$$
\begin{array}{c}
y_1 \cr
y_2 \mathtt{t}_i \cr
y_{3}
\end{array}
$$
```

```
$$\begin{eqnarray}
x' &=& \& x \sin\phi &+& z \cos\phi \\\
z' &=& - \& x \cos\phi &+& z \sin\phi \\\
\end{eqnarray}$$
```

```
$$
x=4
$$
```

Display

$$\begin{aligned} & y_1 \\ & y_2 \mathfrak{t}_i \\ & z_{3,4} \end{aligned}$$

$$\begin{aligned} & y_1 \\ & y_2 \mathfrak{t}_i \\ & y_3 \end{aligned}$$

$$to$$

$$\begin{aligned} x' &= \\ & x \sin \phi \\ & + z \cos \phi \\ z' &= \\ & -x \cos \phi \\ & + z \sin \phi \end{aligned}$$

$$(1.2)$$

$$\begin{aligned} &= \\ &+ \\ - &= \\ &+x \cos \phi \\ &z \sin \phi \end{aligned}$$

$$x = 4$$

1.5 What to do when things go wrong

First, have a look at the common problems listed below. If you can figure it out from these notes, it will be quicker than asking for help.

Check that you have the latest version of any packages that look relevant. Unfortunately it's not always easy to figure out what packages are relevant, but if there was a bug that's already been fixed, it's easy to upgrade and get on with what you wanted to do.

1.5.1 Jupyter fails to start

- Have you `installed it`? ;-)
- If you're using a menu shortcut or Anaconda launcher to start it, try opening a terminal or command prompt and running the command `jupyter nbclassic`.
- If it can't find `jupyter`, you may need to configure your `PATH` environment variable. If you don't know what that means, and don't want to find out, just (re)install Anaconda with the default settings, and it should set up `PATH` correctly.
- If Jupyter gives an error that it can't find `nbclassic`, check with `pip` or `conda` that the `notebook` package is installed.
- Try running `jupyter-nbclassic` (with a hyphen). This should normally be the same as `jupyter nbclassic` (with a space), but if there's any difference, the version with the hyphen is the 'real' launcher, and the other one wraps that.

1.5.2 Jupyter doesn't load or doesn't work in the browser

- Try in another browser (e.g. if you normally use Firefox, try with Chrome). This helps pin down where the problem is.
- Try disabling any browser extensions and/or any Jupyter extensions you have installed.
- Some internet security software can interfere with Jupyter. If you have security software, try turning it off temporarily, and look in the settings for a more long-term solution.
- In the address bar, try changing between `localhost` and `127.0.0.1`. They should be the same, but in some cases it makes a difference.

1.5.3 Jupyter can't start a kernel

Files called *kernel specs* tell Jupyter how to start different kinds of kernels. To see where these are on your system, run `jupyter kernelspec list`:

```
$ jupyter kernelspec list
Available kernels:
python3      /home/takluyver/.local/lib/python3.6/site-packages/ipykernel/resources
bash         /home/takluyver/.local/share/jupyter/kernels/bash
ir           /home/takluyver/.local/share/jupyter/kernels/ir
```

There's a special fallback for the Python kernel: if it doesn't find a real kernelspec, but it can import the `ipykernel` package, it provides a kernel which will run in the same Python environment as the notebook server. A path ending in `ipykernel/resources`, like in the example above, is this default kernel. The default often does what you want,

so if the `python3` kernelspec points somewhere else and you can't start a Python kernel, try deleting or renaming that kernelspec folder to expose the default.

If your problem is with another kernel, not the Python one we maintain, you may need to look for support about that kernel.

1.5.4 Python Environments

Multiple python environments, whether based on Anaconda or Python Virtual environments, are often the source of reported issues. In many cases, these issues stem from the Notebook server running in one environment, while the kernel and/or its resources, derive from another environment. Indicators of this scenario include:

- `import` statements within code cells producing `ImportError` or `ModuleNotFound` exceptions.
- General kernel startup failures exhibited by nothing happening when attempting to execute a cell.

In these situations, take a close look at your environment structure and ensure all packages required by your notebook's code are installed in the correct environment. If you need to run the kernel from different environments than your Notebook server, check out [IPython's documentation](#) for using kernels from different environments as this is the recommended approach. Anaconda's `nb_conda_kernels` package might also be an option for you in these scenarios.

Another thing to check is the `kernel.json` file that will be located in the aforementioned *kernel specs* directory identified by running `jupyter kernelspec list`. This file will contain an `argv` stanza that includes the actual command to run when launching the kernel. Oftentimes, when reinstalling python environments, a previous `kernel.json` will reference an python executable from an old or non-existent location. As a result, it's always a good idea when encountering kernel startup issues to validate the `argv` stanza to ensure all file references exist and are appropriate.

1.5.5 Windows Systems

Although Jupyter Notebook is primarily developed on the various flavors of the Unix operating system it also supports Microsoft Windows - which introduces its own set of commonly encountered issues, particularly in the areas of security, process management and lower-level libraries.

pywin32 Issues

The primary package for interacting with Windows' primitives is `pywin32`.

- Issues surrounding the creation of the kernel's communication file utilize `jupyter_core`'s `secure_write()` function. This function ensures a file is created in which only the owner of the file has access. If libraries like `pywin32` are not properly installed, issues can arise when it's necessary to use the native Windows libraries.

Here's a portion of such a traceback:

```
File "c:\users\jovyan\python\myenv.venv\lib\site-packages\jupyter_core\paths.py", line 424, in secure_write
win32_restrict_file_to_user(fname)
File "c:\users\jovyan\python\myenv.venv\lib\site-packages\jupyter_core\paths.py", line 359, in win32_restrict_file_to_user
import win32api
ImportError: DLL load failed: The specified module could not be found.
```

- As noted earlier, the installation of `pywin32` can be problematic on Windows configurations. When such an issue occurs, you may need to revisit how the environment was setup. Pay careful attention to whether you're running the 32 or 64 bit versions of Windows and be sure to install appropriate packages for that environment.

Here's a portion of such a traceback:

```
File "C:\Users\jovyan\AppData\Roaming\Python\Python37\site-packages\jupyter_core\
↳paths.py", line 435, in secure_write
win32_restrict_file_to_user(fname)
File "C:\Users\jovyan\AppData\Roaming\Python\Python37\site-packages\jupyter_core\
↳paths.py", line 361, in win32_restrict_file_to_user
import win32api
ImportError: DLL load failed: %1 is not a valid Win32 application
```

Resolving pywin32 Issues

In this case, your pywin32 module may not be installed correctly and the following should be attempted:

```
pip install --upgrade pywin32
```

or:

```
conda install --force-reinstall pywin32
```

followed by:

```
python.exe Scripts/pywin32_postinstall.py -install
```

where Scripts is located in the active Python's installation location.

- Another common failure specific to Windows environments is the location of various python commands. On *nix systems, these typically reside in the bin directory of the active Python environment. However, on Windows, these tend to reside in the Scripts folder - which is a sibling to bin. As a result, when encountering kernel startup issues, again, check the argv stanza and verify it's pointing to a valid file. You may find that it's pointing in bin when Scripts is correct, or the referenced file does not include its .exe extension - typically resulting in FileNotFoundError exceptions.

1.5.6 This Worked An Hour Ago

The Jupyter stack is very complex and rightfully so, there's a lot going on. On occasion you might find the system working perfectly well, then, suddenly, you can't get past a certain cell due to import failures. In these situations, it's best to ask yourself if any new python files were added to your notebook development area.

These issues are usually evident by carefully analyzing the traceback produced in the notebook error or the Notebook server's command window. In these cases, you'll typically find the Python kernel code (from IPython and ipykernel) performing *its* imports and notice a file from your Notebook development error included in that traceback followed by an AttributeError:

```
File "C:\Users\jovyan\anaconda3\lib\site-packages\ipykernel\connect.py", line 13, in
from IPython.core.profiledir import ProfileDir
File "C:\Users\jovyan\anaconda3\lib\site-packages\IPython_init.py", line 55, in
from .core.application import Application
...
File "C:\Users\jovyan\anaconda3\lib\site-packages\ipython_genutils\path.py", line 13, in
import random
File "C:\Users\jovyan\Desktop\Notebooks\random.py", line 4, in
rand_set = random.sample(english_words_lower_set, 12)
AttributeError: module 'random' has no attribute 'sample'
```

What has happened is that you have named a file that conflicts with an installed package that is used by the kernel software and now introduces a conflict preventing the kernel's startup.

Resolution: You'll need to rename your file. A best practice would be to prefix or *namespace* your files so as not to conflict with any python package.

1.5.7 Asking for help

As with any problem, try searching to see if someone has already found an answer. If you can't find an existing answer, you can ask questions at:

- The [Jupyter Discourse Forum](#)
- The [jupyter-notebook](#) tag on [Stackoverflow](#)
- Peruse the [jupyter/help](#) repository on [Github](#) (read-only)
- Or in an issue on another repository, if it's clear which component is responsible. Typical repositories include:
 - [jupyter_core](#) - `secure_write()` and file path issues
 - [jupyter_client](#) - kernel management issues found in Notebook server's command window.
 - [IPython](#) and [ipykernel](#) - kernel runtime issues typically found in Notebook server's command window and/or Notebook cell execution.

Gathering Information

Should you find that your problem warrants that an issue be opened in [notebook](#) please don't forget to provide details like the following:

- What error messages do you see (within your notebook and, more importantly, in the Notebook server's command window)?
- What platform are you on?
- How did you install Jupyter?
- What have you tried already?

The `jupyter troubleshoot` command collects a lot of information about your installation, which can also be useful.

When providing textual information, it's most helpful if you can *scrape* the contents into the issue rather than providing a screenshot. This enables others to select pieces of that content so they can search more efficiently and try to help.

Remember that it's not anyone's job to help you. We want Jupyter to work for you, but we can't always help everyone individually.

1.6 Changelog

1.6.1 1.0.0

([Full Changelog](#))

Bugs fixed

- Fix typo in documentation #249 (@echarles)

Documentation improvements

- Fix typo in documentation #249 (@echarles)

Contributors to this release

(GitHub contributors page for this release)

@echarles

1.6.2 0.5.6

(Full Changelog)

Enhancements made

- Add show_banner trait to control the banner display #243 (@echarles)

Other merged PRs

- Updated main page README. #246 (@eric snekbytes)
- Use the correct icon to display the application #245 (@befeleme)
- State the correct name of the application in the desktop entry #244 (@befeleme)

Contributors to this release

(GitHub contributors page for this release)

@befeleme | @echarles | @eric snekbytes

1.6.3 0.5.5

(Full Changelog)

Merged PRs

- Remove redirect from /notebooks to /files #235 (@mwouts)

Contributors to this release

(GitHub contributors page for this release)

@echarles | @matthew-brett | @mwouts

1.6.4 0.5.4

(Full Changelog)

Bugs fixed

- Copy marked.umd.js to marked.js and revert loading #236 (@echarles)
- CI Test Fixed Round6a #234 (@eric snekbytes)
- CI Test Fixes Round 5 #232 (@eric snekbytes)

Maintenance and upkeep improvements

- CI Test Fixed Round6a #234 (@eric snekbytes)
- CI Test Fixes Round 5 #232 (@eric snekbytes)

Contributors to this release

(GitHub contributors page for this release)

@echarles | @eric snekbytes

1.6.5 0.5.3

(Full Changelog)

Bugs fixed

- Fix generating of .mo and .json files for nl translation #221 (@frenzymadness)
- Fix banner icons #220 (@echarles)

Maintenance and upkeep improvements

- CI Test Fixes Round 4 #226 (@eric snekbytes)

Contributors to this release

(GitHub contributors page for this release)

@echarles | @eric snekbytes | @frenzymadness | @RRosio

1.6.6 0.5.2

(Full Changelog)

Enhancements made

- Add news banner and log #215 (@echarles)

Bugs fixed

- CI Test Fixes Round 3 #216 (@eric snekbytes)
- CI Test Fixes Round 2 #212 (@eric snekbytes)
- Update get_rendered_contents further marked javascript upgrade #210 (@echarles)

Maintenance and upkeep improvements

- CI Test Fixes Round 3 #216 (@eric snekbytes)
- CI Test Fixes Round 2 #212 (@eric snekbytes)

Documentation improvements

- Update security concerns link #218 (@RRosio)

Other merged PRs

- Don't require two different minimal versions of jupyter_server #208 (@hroncok)

Contributors to this release

(GitHub contributors page for this release)

@echarles | @ellisonbg | @eric snekbytes | @hroncok | @RRosio

1.6.7 0.5.1

(Full Changelog)

Bugs fixed

- Add packages and package_data back in the setup_args #206 (@echarles)
- Add static and templates as artifact for hatch build #205 (@echarles)

Contributors to this release

(GitHub contributors page for this release)

@echarles

1.6.8 0.5.0

(Full Changelog)

Enhancements made

- Upgrade to marked 4.0.10 #201 (@echarles)

Bugs fixed

- Ci test fixes round1 #202 (@eric snekbytes)
- fix loading of static and template path config #192 (@minrk)
- Add pytest_jupyter to the list of test deps #190 (@echarles)
- Align setup.py and setup.cfg #181 (@echarles)

Maintenance and upkeep improvements

- Ci test fixes round1 #202 (@eric snekbytes)
- Remove duplicate notebook_shim from setup.cfg #197 (@frenzymadness)
- Upgrade moment javascript package #186 (@echarles)
- Add dependabot config #175 (@blink1073)

Documentation improvements

- Update documentation #196 (@RRosio)
- Update the project URL #194 (@jtpio)

Other merged PRs

- Fix the jupyter server base.handlers import #187 (@echarles)
- Revert pull request 168 #184 (@echarles)

Contributors to this release

(GitHub contributors page for this release)

@blink1073 | @echarles | @ericsnekbytes | @frenzymadness | @jtpio | @kevin-bates | @minrk | @RRosio

1.6.9 0.4.8

(Full Changelog)

Enhancements made

- Playwright Testing Conversion #170 (@ericsnekbytes)

Bugs fixed

- Handle async and sync contents managers #172 (@blink1073)

Maintenance and upkeep improvements

- Playwright port #139 (@ericsnekbytes)

Documentation improvements

- Documentation for the extensions configurator #174 (@echarles)
- Updating contributing docs for playwright tests #171 (@RRosio)

Other merged PRs

- Update docs to remove install command #173 (@dleen)

Contributors to this release

(GitHub contributors page for this release)

@blink1073 | @dleen | @echarles | @ericsnekbytes | @RRosio

1.6.10 0.4.7

(Full Changelog)

New features added

- Fix/nbclassic base url #168 (@mcrutch)

Enhancements made

- Add a redirect from /nbclassic to /nbclassic/tree if both notebook>=7 if both notebook>=7 and nbclassic are installed #166 (@echarles)

Bugs fixed

- Allow NBClassic to work with NB7 and ServerApp.base_url #165 (@mcrutch)

Contributors to this release

(GitHub contributors page for this release)

@echarles | @mcrutch

1.6.11 0.4.6

(Full Changelog)

New features added

- Adds new file custom-preload that is loaded in a different order than custom.js before the main.js script is executed #155 (@lneves12)

Bugs fixed

- Use var instead of const to ensure js-test succeed #159 (@echarles)

Documentation improvements

- add badge for documentation #160 (@RRosio)

Contributors to this release

(GitHub contributors page for this release)

@echarles | @Ineves12 | @RRosio

1.6.12 0.4.5

(Full Changelog)

Bugs fixed

- fix: display the correct about box if jupyter notebook command is used #156 (@echarles)

Contributors to this release

(GitHub contributors page for this release)

@echarles | @Ineves12

1.6.13 0.4.4

(Full Changelog)

Enhancements made

- Stop looking in the old IPython directory for nbextensions #153 (@minrk)
- Handlers under nbclassic if notebook 7 is found #141 (@echarles)

Bugs fixed

- Fix resource display when redirecting from /notebooks/* #148 (@echarles)
- Rename duplicate entrypoints #138 (@echarles)
- Point nbclassic at the classic jupyter_notebook_config path #137 (@Zsailer)

Maintenance and upkeep improvements

- Updating JS Dependencies #152 (@RRosio)
- Get the notebook version from notebook._version #133 (@echarles)

Documentation improvements

- Updating documentation for nbclassic #134 (@RRosio)

Other merged PRs

- UI: About box to display nbclassic instead of notebook #147 (@echarles)

Contributors to this release

(GitHub contributors page for this release)

@echarles | @jtpio | @minrk | @RRosio | @Zsailer

1.6.14 0.4.3

(Full Changelog)

Enhancements made

- Handle execution errors with empty traceback entries similar to Lab #126 (@kevin-bates)

Bugs fixed

- fix some import errors in 0.4 #130 (@minrk)
- fix enable-on-install #127 (@minrk)

Maintenance and upkeep improvements

- Fix handling of dev version when releasing #131 (@echarles)
- Update release notes for 0.4.2 #125 (@echarles)

Other merged PRs

- Fix notebook shim version check #129 (@minrk)

Contributors to this release

(GitHub contributors page for this release)

@echarles | @kevin-bates | @minrk

1.6.15 0.4.2

(Full Changelog)

Enhancements made

- Don't shim notebook if notebook<7 is available #123 (@echarles)

Bugs fixed

- Relax assert for IPythonHandler in case of notebook<6 #122 (@echarles)

Other merged PRs

- Fix executable name in desktop file #119 (@antonio-rojas)

Contributors to this release

(GitHub contributors page for this release)

@antonio-rojas | @echarles

1.6.16 0.4.1

This release has been skipped by the release process.

1.6.17 0.4.0

(Full Changelog)

Enhancements made

- Shim to support the notebook extensions #113 (@echarles)
- Fix Check Release / check_release CI job #102 (@echarles)
- nbclassic to use its own static assets #96 (@echarles)

Bugs fixed

- [HOT FIX] Test Selenium CI #114 (@echarles)
- Fix link_check CI Job #111 (@echarles)
- Fix downstream CI Job #110 (@echarles)
- Disable Test Minimum Version CI Job #109 (@echarles)
- Fix Make SDist CI Job by installing babel as a build dependency #108 (@RRosio)
- Exclude Services tests from Linux JS Tests Job and Separate Workflow for flaky Selenium Tests #105 (@RRosio)
- CI fix for “Testing nbclassic” on macos 3.7, 3.8, 3.9, 3.10 #104 (@eric snekbytes)
- Fix part of the Linux JS Tests / Notebook and Base Test Failures in CI Job #103 (@RRosio)

Maintenance and upkeep improvements

- Add the notebook 6.4.x tests, docs and companion files (with git history) #94 (@echarles)
- Add the notebook static assets (with git history) #93 (@echarles)

Other merged PRs

- Add post-version-spec in tool.jupyter-releaser.options #116 (@echarles)
- CI fix for “Testing nbclassic” on macos / pypy #106 (@eric snekbytes)
- [CI] Fix the Docs Tests job #99 (@echarles)

Contributors to this release

(GitHub contributors page for this release)

@blink1073 | @echarles | @eric snekbytes | @RRosio | @Zsailer

1.6.18 0.3.7

(Full Changelog)

Maintenance and upkeep improvements

- Clean up packaging and CI #91 (@blink1073)

Contributors to this release

(GitHub contributors page for this release)

@blink1073

1.6.19 0.3.6

(Full Changelog)

Maintenance and upkeep improvements

- Depend on notebook_shim package for server-side shim layer #88 (@Zsailer)

Contributors to this release

(GitHub contributors page for this release)

@jtpio | @Zsailer

1.6.20 0.3.5

(Full Changelog)

Bugs fixed

- ensure extension name is passed to TerminalHandler #79 (@minrk)

Maintenance and upkeep improvements

- Clean up downstream tests #82 (@blink1073)
- Enforce labels on PRs #80 (@blink1073)

Contributors to this release

(GitHub contributors page for this release)

@blink1073 | @minrk

1.6.21 0.3.4

(Full Changelog)

Bugs fixed

- Monkeypatch IPythonHandler to find nbclassic's jinja templates #77 (@Zsailer)

Maintenance and upkeep improvements

- Run JupyterLab browser tests as downstream tests #76 (@Zsailer)

Other merged PRs

- add check-release workflow #75 (@Zsailer)

Contributors to this release

(GitHub contributors page for this release)

@Zsailer

1.6.22 0.3.3

(Full Changelog)

Maintenance and upkeep improvements

- Use Jupyter Packaging and other cleanup #68 (@Zsailer)

Other merged PRs

- add missing lines to manifest #72 (@Zsailer)
- add changelog comments for jupyter-releaser #70 (@Zsailer)
- add setup.py back to enable jupyter_releaser #69 (@Zsailer)
- Add workflow to test JupyterLab #67 (@Zsailer)
- Expose classic notebook's static assets from their original endpoints #63 (@Zsailer)

Contributors to this release

(GitHub contributors page for this release)

@meeseeksmachine | @Zsailer

1.6.23 0.3.2

(Full Changelog)

Merged PRs

- ExtensionManager's link_extension changed call signature #64 (@athornton)
- Fix link to the Jupyter Notebook repo in README #62 (@saiwing-yeung)

Contributors to this release

(GitHub contributors page for this release)

@athornton | @saiwing-yeung | @welcome

1.6.24 0.3.1

(Full Changelog)

Maintenance and upkeep improvements

- bump jupyter_server dependency to 1.8 #58 (@Zsailer)

Contributors to this release

(GitHub contributors page for this release)

@Zsailer

1.6.25 0.3.0

(Full Changelog)

Enhancements made

- Support creating terminal with a given name. #52 (@cailiang9)

Bugs fixed

- BUG fix: correct redirection to {base_url}/edit/*. #55 (@cailiang9)

Maintenance and upkeep improvements

- patch server's sorted_extensions to prioritize nbclassic #56 (@Zsailer)

Contributors to this release

(GitHub contributors page for this release)

@cailiang9 | @welcome | @Zsailer

1.6.26 0.2.8

(Full Changelog)

Merged PRs

- Remove forced sorting of extensions #49 (@minrk)
- Add Changelog #48 (@blink1073)

Contributors to this release

(GitHub contributors page for this release)

@blink1073 | @minrk | @welcome

1.6.27 0.2.7

1.6.28 Merged PRs

- Fix deprecation warning when importing jupyter_server.transutils._ #47 (@martinRenou)
- Add a redirect handler to open non-notebook files from the cli #45 (@jtpio)
- Add default_url trait to NotebookApp #42 (@afshin)
- Fix GitHub Actions badge #40 (@jtpio)
- Run jupyter nbclassic -h on CI #29 (@jtpio)

1.6.29 Contributors to this release

(GitHub contributors page for this release)

@afshin | @blink1073 | @jtpio | @martinRenou | @welcome | @Zsailer

1.6.30 0.2.6

(Full Changelog)

Merged PRs

- Ignore some traits in shim layer #38 (@afshin)

Contributors to this release

(GitHub contributors page for this release)

@afshin

1.6.31 0.2.5

(Full Changelog)

Merged PRs

- Update jupyter_server version, update to use prefixed fixtures #37 (@afshin)

Contributors to this release

(GitHub contributors page for this release)

@afshin

1.6.32 0.2.4

(Full Changelog)

Merged PRs

- Exclude tests from dist #36 (@bollwyvl)
- Update release instructions #34 (@jasongrout)

Contributors to this release

(GitHub contributors page for this release)

@bollwyvl | @jasongrout | @Zsailer

1.6.33 0.2.3

(Full Changelog)

Merged PRs

- Moves terminal websocket handling back to Jupyter Server #33 (@jasongrout)

Contributors to this release

(GitHub contributors page for this release)

@jasongrout

1.6.34 0.2.2

(Full Changelog)

Merged PRs

- Add terminal and editor handlers #31 (@afshin)

Contributors to this release

(GitHub contributors page for this release)

@afshin

1.6.35 0.2.1

(Full Changelog)

Merged PRs

- Add setupbase.py to MANIFEST.in #28 (@jtpio)
- Add LICENSE #27 (@jtpio)

Contributors to this release

(GitHub contributors page for this release)

@jtpio

1.7 Comms

Comms allow custom messages between the frontend and the kernel. They are used, for instance, in `ipywidgets` to update widget state.

A comm consists of a pair of objects, in the kernel and the frontend, with an automatically assigned unique ID. When one side sends a message, a callback on the other side is triggered with that message data. Either side, the frontend or kernel, can open or close the comm.

See also:

Custom Messages

The messaging specification section on comms

1.7.1 Opening a comm from the kernel

First, the function to accept the comm must be available on the frontend. This can either be specified in a *requires* module, or registered in a registry, for example when an *extension* is loaded. This example shows a frontend comm target registered in a registry:

```
Jupyter.notebook.kernel.comm_manager.register_target('my_comm_target',
    function(comm, msg) {
        // comm is the frontend comm instance
        // msg is the comm_open message, which can carry data

        // Register handlers for later messages:
        comm.on_msg(function(msg) {...});
        comm.on_close(function(msg) {...});
        comm.send({'foo': 0});
    });
```

Now that the frontend comm is registered, you can open the comm from the kernel:

```
from ipykernel.comm import Comm

# Use comm to send a message from the kernel
my_comm = Comm(target_name='my_comm_target', data={'foo': 1})
my_comm.send({'foo': 2})

# Add a callback for received messages.
@my_comm.on_msg
def _recv(msg):
    # Use msg['content']['data'] for the data in the message
```

This example uses the IPython kernel; it's up to each language kernel what API, if any, it offers for using comms.

1.7.2 Opening a comm from the frontend

This is very similar to above, but in reverse. First, a comm target must be registered in the kernel. For instance, this may be done by code displaying output: it will register a target in the kernel, and then display output containing Javascript to connect to it.

```
def target_func(comm, open_msg):
    # comm is the kernel Comm instance
    # msg is the comm_open message

    # Register handler for later messages
    @comm.on_msg
    def _recv(msg):
        # Use msg['content']['data'] for the data in the message
        comm.send({'echo': msg['content']['data']})

    # Send data to the frontend on creation
    comm.send({'foo': 5})

get_ipython().kernel.comm_manager.register_target('my_comm_target', target_func)
```

This example uses the IPython kernel again; this example will be different in other kernels that support comms. Refer to the specific language kernel's documentation for comms support.

And then open the comm from the frontend:

```
const comm = Jupyter.notebook.kernel.comm_manager.new_comm('my_comm_target', {'foo': 6})
// Send data
comm.send({'foo': 7})

// Register a handler
comm.on_msg(function(msg) {
    console.log(msg.content.data.foo);
});
```

Information about using and troubleshooting NbClassic as well as examples and changes throughout the history of the project.

CONFIGURATION

2.1 Configuration Overview

As NbClassic is a Jupyter Server Extension that is intended as an intermediary project while users migrate from Notebook 6 to Notebook 7, this document makes references to configuration files carried over from the classic Jupyter Notebook which will be executed when running nbclassic. For more general documentation regarding the Jupyter Server that NbClassic uses please visit the [Jupyter Server documentation](#).

Beyond the default configuration settings, you can configure a rich array of options to suit your workflow. Here are areas that are commonly configured when using Jupyter NbClassic:

- *Jupyter's common configuration system*
- *NbClassic server*
- *NbClassic front-end client*
- *Notebook extensions*

Let's look at highlights of each area.

2.1.1 Jupyter's Common Configuration system

Jupyter applications, from the Notebook to JupyterHub to nbgrader, share a common configuration system. The process for creating a configuration file and editing settings is similar for all the Jupyter applications.

- [Jupyter's Common Configuration Approach](#)
- [Common Directories and File Locations](#)
- [Language kernels](#)
- [traitlets](#) provide a low-level architecture for configuration.

2.1.2 NbClassic Server

The NbClassic server runs the language kernel and communicates with the front-end NbClassic client (i.e. the familiar notebook interface).

- Configuring the NbClassic server

To create a `jupyter_notebook_config.py` file in the `.jupyter` directory, with all the defaults commented out, use the following command:

```
$ jupyter nbclassic --generate-config
```

Command line arguments for configuration settings are documented in the configuration file and the user documentation.

- Review the section: [Running a Notebook server](#)
- Related: [Configuring a language kernel](#) to run in the Notebook server enables your server to run other languages, like R or Julia.

2.1.3 NbClassic front-end client

Configuring the NbClassic frontend

Note: The ability to configure the nbclassic frontend UI and preferences is still a work in progress.

This document is a rough explanation on how you can persist some configuration options for the nbclassic JavaScript. There is no exhaustive list of all the configuration options as most options are passed down to other libraries, which means that non valid configuration can be ignored without any error messages.

How front end configuration works

The frontend configuration system works as follows:

- get a handle of a configurable JavaScript object.
- access its configuration attribute.
- update its configuration attribute with a JSON patch.

Example - Changing the notebook's default indentation

This example explains how to change the default setting `indentUnit` for CodeMirror Code Cells:

```
var cell = Jupyter.notebook.get_selected_cell();
var config = cell.config;
var patch = {
  CodeCell:{
    cm_config:{indentUnit:2}
  }
}
config.update(patch)
```

You can enter the previous snippet in your browser's JavaScript console once. Then reload the notebook page in your browser. Now, the preferred indent unit should be equal to two spaces. The custom setting persists and you do not need to reissue the patch on new notebooks.

`indentUnit`, used in this example, is one of the many [CodeMirror options](#) which are available for configuration.

You can similarly change the options of the file editor by entering the following snippet in the browser's Javascript console once (from a file editing page):.

```
var config = Jupyter.editor.config
var patch = {
  Editor: {
    codemirror_options: {
      indentUnit: 2
    }
  }
}
config.update(patch)
```

Example - Restoring the notebook's default indentation

If you want to restore a notebook frontend preference to its default value, you will enter a JSON patch with a null value for the preference setting.

For example, let's restore the indent setting `indentUnit` to its default of four spaces. Enter the following code snippet in your JavaScript console:

```
var cell = Jupyter.notebook.get_selected_cell();
var config = cell.config;
var patch = {
  CodeCell:{
    cm_config:{indentUnit: null} // only change here.
  }
}
config.update(patch)
```

Reload the notebook in your browser and the default indent should again be two spaces.

Persisting configuration settings

Under the hood, Jupyter will persist the preferred configuration settings in `~/.jupyter/nbconfig/<section>.json`, with `<section>` taking various value depending on the page where the configuration is issued. `<section>` can take various values like `notebook`, `tree`, and `editor`. A `common` section contains configuration settings shared by all pages.

Persisting configuration settings

A banner might be shown to users to inform them about news or updates. This banner can be hidden launching the server with the `show_banner` trait.:

```
jupyter notebook --NotebookApp.show_banner=False
```

2.1.4 Notebook extensions

- [Distributing Jupyter Extensions as Python Packages](#)
- [Extending the Notebook](#)

Security in Jupyter notebooks: Since security policies vary from organization to organization, we encourage you to consult with your security team on settings that would be best for your use cases. Our documentation offers some responsible security practices, and we recommend becoming familiar with the practices.

2.2 Config file and command line options

The notebook server can be run with a variety of command line arguments. A list of available options can be found below in the *options section*.

Defaults for these options can also be set by creating a file named `jupyter_notebook_config.py` in your Jupyter folder. The Jupyter folder is in your home directory, `~/.` `jupyter`.

To create a `jupyter_notebook_config.py` file, with all the defaults commented out, you can use the following command line:

```
$ jupyter notebook --generate-config
```

2.2.1 Options

This list of options can be generated by running the following and hitting enter:

```
$ jupyter notebook --help
```

Application.log_datefmt

[Unicode] Default: '%Y-%m-%d %H:%M:%S'

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

Application.log_format

[Unicode] Default: '%[(name)s]%(highlevel)s %(message)s'

The Logging format template

Application.log_level

[any of `0` | `10` | `20` | `30` | `40` | `50` | `'DEBUG'` | `'INFO'` | `'WARN'` | `'ERROR'` | `'CRITICAL'`]

Default: `30`

Set the log level by value or name.

Application.logging_config

[Dict] Default: `{}`

Configure additional log handlers.

The default stderr logs handler is configured by the `log_level`, `log_datefmt` and `log_format` settings.

This configuration can be used to configure additional handlers (e.g. to output the log to a file) or for finer control over the default handlers.

If provided this should be a logging configuration dictionary, for more information see: <https://docs.python.org/3/library/logging.config.html#logging-config-dictschema>

This dictionary is merged with the base logging configuration which defines the following:

- A logging formatter intended for interactive use called `console`.
- A logging handler that writes to `stderr` called `console` which uses the formatter `console`.
- A logger with the name of this application set to `DEBUG` level.

This example adds a new handler that writes to a file:

```
c.Application.logging_config = {
    'handlers': {
        'file': {
            'class': 'logging.FileHandler',
            'level': 'DEBUG',
            'filename': '<path/to/file>',
        }
    },
    'loggers': {
        '<application-name>': {
            'level': 'DEBUG',
            # NOTE: if you don't list the default "console"
            # handler here then it will be disabled
            'handlers': ['console', 'file'],
        },
    },
}
```

Application.show_config

[Bool] Default: False

Instead of starting the Application, dump configuration to stdout

Application.show_config_json

[Bool] Default: False

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

JupyterApp.answer_yes

[Bool] Default: False

Answer yes to any prompts.

JupyterApp.config_file

[Unicode] Default: ''

Full path of a config file.

JupyterApp.config_file_name

[Unicode] Default: ''

Specify a config file to load.

JupyterApp.generate_config

[Bool] Default: False

Generate default config file.

JupyterApp.log_datefmt

[Unicode] Default: '%Y-%m-%d %H:%M:%S'

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

JupyterApp.log_format

[Unicode] Default: '%(name)s %(levelname)s %(message)s'

The Logging format template

JupyterApp.log_level

[any of 0 ``| ``10 ``| ``20 ``| ``30 ``| ``40 ``| ``50 ``| 'DEBUG' | 'INFO' | 'WARN' | 'ERROR' | 'CRITICAL' ``]

Default: 30

Set the log level by value or name.

JupyterApp.logging_config

[Dict] Default: {}

Configure additional log handlers.

The default stderr logs handler is configured by the log_level, log_datefmt and log_format settings.

This configuration can be used to configure additional handlers (e.g. to output the log to a file) or for finer control over the default handlers.

If provided this should be a logging configuration dictionary, for more information see: <https://docs.python.org/3/library/logging.config.html#logging-config-dictschema>

This dictionary is merged with the base logging configuration which defines the following:

- A logging formatter intended for interactive use called `console`.
- A logging handler that writes to stderr called `console` which uses the formatter `console`.
- A logger with the name of this application set to DEBUG level.

This example adds a new handler that writes to a file:

```
c.Application.logging_config = {
    'handlers': {
        'file': {
            'class': 'logging.FileHandler',
            'level': 'DEBUG',
            'filename': '<path/to/file>',
        },
    },
    'loggers': {
        '<application-name>': {
            'level': 'DEBUG',
            # NOTE: if you don't list the default "console"
            # handler here then it will be disabled
            'handlers': ['console', 'file'],
        },
    },
}
```

JupyterApp.show_config

[Bool] Default: False

Instead of starting the Application, dump configuration to stdout

JupyterApp.show_config_json

[Bool] Default: False

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

ExtensionApp.answer_yes

[Bool] Default: False

Answer yes to any prompts.

ExtensionApp.config_file

[Unicode] Default: ''

Full path of a config file.

ExtensionApp.config_file_name

[Unicode] Default: ''

Specify a config file to load.

ExtensionApp.default_url

[Unicode] Default: ''

No description

ExtensionApp.generate_config

[Bool] Default: False

Generate default config file.

ExtensionApp.handlers

[List] Default: []

Handlers appended to the server.

ExtensionApp.log_datefmt

[Unicode] Default: '%Y-%m-%d %H:%M:%S'

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

ExtensionApp.log_format

[Unicode] Default: '%[(name)s]%(highlevel)s %(message)s'

The Logging format template

ExtensionApp.log_level

[any of 0 `` | `` 10 `` | `` 20 `` | `` 30 `` | `` 40 `` | `` 50 `` | 'DEBUG' | 'INFO' | 'WARN' | 'ERROR' | 'CRITICAL']
Default: 30

Set the log level by value or name.

ExtensionApp.logging_config

[Dict] Default: {}

Configure additional log handlers.

The default stderr logs handler is configured by the log_level, log_datefmt and log_format settings.

This configuration can be used to configure additional handlers (e.g. to output the log to a file) or for finer control over the default handlers.

If provided this should be a logging configuration dictionary, for more information see: <https://docs.python.org/3/library/logging.config.html#logging-config-dictschema>

This dictionary is merged with the base logging configuration which defines the following:

- A logging formatter intended for interactive use called `console`.
- A logging handler that writes to stderr called `console` which uses the formatter `console`.
- A logger with the name of this application set to `DEBUG` level.

This example adds a new handler that writes to a file:

```
c.Application.logging_config = {
    'handlers': {
        'file': {
            'class': 'logging.FileHandler',
            'level': 'DEBUG',
            'filename': '<path/to/file>',
        }
    },
    'loggers': {
        '<application-name>': {
            'level': 'DEBUG',
            # NOTE: if you don't list the default "console"
            # handler here then it will be disabled
            'handlers': ['console', 'file'],
        },
    },
}
```

ExtensionApp.open_browser

[Bool] Default: False

Whether to open in a browser after starting.

The specific browser used is platform dependent and determined by the python standard library *webbrowser* module, unless it is overridden using the `--browser` (`ServerApp.browser`) configuration option.

ExtensionApp.settings

[Dict] Default: {}

Settings that will be passed to the server.

ExtensionApp.show_config

[Bool] Default: False

Instead of starting the Application, dump configuration to stdout

ExtensionApp.show_config_json

[Bool] Default: False

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

ExtensionApp.static_paths

[List] Default: []

paths to search for serving static files.

This allows adding javascript/css to be available from the notebook server machine, or overriding individual files in the IPython

ExtensionApp.static_url_prefix

[Unicode] Default: ''

Url where the static assets for the extension are served.

ExtensionApp.template_paths

[List] Default: []

Paths to search for serving jinja templates.

Can be used to override templates from `notebook.templates`.

NotebookApp.answer_yes
[Bool] Default: False

Answer yes to any prompts.

NotebookApp.config_file
[Unicode] Default: ''

Full path of a config file.

NotebookApp.config_file_name
[Unicode] Default: 'jupyter_notebook_config'

Specify a config file to load.

NotebookApp.default_url
[Unicode] Default: '/tree'

No description

NotebookApp.enable_mathjax
[Bool] Default: True

Whether to enable MathJax for typesetting math/TeX

MathJax is the javascript library Jupyter uses to render math/LaTeX. It is very large, so you may want to disable it if you have a slow internet connection, or for offline use of the notebook.

When disabled, equations etc. will appear as their untransformed TeX source.

NotebookApp.extra_nbextensions_path
[List] Default: []

extra paths to look for Javascript notebook extensions

NotebookApp.extra_static_paths
[List] Default: []

Extra paths to search for serving static files.

This allows adding javascript/css to be available from the notebook server machine, or overriding individual files in the IPython

NotebookApp.extra_template_paths
[List] Default: []

Extra paths to search for serving jinja templates.

Can be used to override templates from notebook.templates.

NotebookApp.generate_config
[Bool] Default: False

Generate default config file.

NotebookApp.handlers
[List] Default: []

Handlers appended to the server.

NotebookApp.ignore_minified_js
[Bool] Default: False

Deprecated: Use minified JS file or not, mainly use during dev to avoid JS recompilation

NotebookApp.jinja2_options

[Dict] Default: {}

Options to pass to the jinja2 environment for this

NotebookApp.jinja_environment_options

[Dict] Default: {}

Supply extra arguments that will be passed to Jinja environment.

NotebookApp.jinja_template_vars

[Dict] Default: {}

Extra variables to supply to jinja templates when rendering.

NotebookApp.log_datefmt

[Unicode] Default: '%Y-%m-%d %H:%M:%S'

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

NotebookApp.log_format

[Unicode] Default: '%[(name)s]%(highlevel)s %(message)s'

The Logging format template

NotebookApp.log_level

[any of 0 ``|`` 10 ``|`` 20 ``|`` 30 ``|`` 40 ``|`` 50 ``|`` 'DEBUG' | 'INFO' | 'WARN' | 'ERROR' | 'CRITICAL' ``]`
Default: 30

Set the log level by value or name.

NotebookApp.logging_config

[Dict] Default: {}

Configure additional log handlers.

The default stderr logs handler is configured by the log_level, log_datefmt and log_format settings.

This configuration can be used to configure additional handlers (e.g. to output the log to a file) or for finer control over the default handlers.

If provided this should be a logging configuration dictionary, for more information see: <https://docs.python.org/3/library/logging.config.html#logging-config-dictschema>

This dictionary is merged with the base logging configuration which defines the following:

- A logging formatter intended for interactive use called `console`.
- A logging handler that writes to stderr called `console` which uses the formatter `console`.
- A logger with the name of this application set to `DEBUG` level.

This example adds a new handler that writes to a file:

```
c.Application.logging_config = {
    'handlers': {
        'file': {
            'class': 'logging.FileHandler',
            'level': 'DEBUG',
            'filename': '<path/to/file>',
        }
    },
    'loggers': {
        '<application-name>': {
```

(continues on next page)

(continued from previous page)

```

        'level': 'DEBUG',
        # NOTE: if you don't list the default "console"
        # handler here then it will be disabled
        'handlers': ['console', 'file'],
    },
}
}

```

NotebookApp.mathjax_config

[Unicode] Default: 'TeX-AMS-MML_HTMLorMML-full, Safe'

The MathJax.js configuration file that is to be used.

NotebookApp.mathjax_url

[Unicode] Default: ''

A custom url for MathJax.js.

Should be in the form of a case-sensitive url to MathJax, for example:
 /static/components/MathJax/MathJax.js

NotebookApp.nbserver_extensions

[Dict] Default: {}

Dict of Python modules to load as notebook server extensions. Entry values can be used to enable and disable the loading of the extensions. The extensions will be loaded in alphabetical order.

NotebookApp.open_browser

[Bool] Default: True

Whether to open in a browser after starting.

The specific browser used is platform dependent and determined by the python standard library *webbrowser* module, unless it is overridden using the `--browser` (ServerApp.browser) configuration option.

NotebookApp.quit_button

[Bool] Default: True

If True, display a button in the dashboard to quit
 (shutdown the notebook server).

NotebookApp.settings

[Dict] Default: {}

Settings that will be passed to the server.

NotebookApp.show_banner

[Bool] Default: True

Whether the banner is displayed on the page.

By default, the banner is displayed.

NotebookApp.show_config

[Bool] Default: False

Instead of starting the Application, dump configuration to stdout

NotebookApp.show_config_json

[Bool] Default: False

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

NotebookApp.static_url_prefix

[Unicode] Default: ''

Url where the static assets for the extension are served.

ServerApp.allow_credentials

[Bool] Default: False

Set the Access-Control-Allow-Credentials: true header

ServerApp.allow_origin

[Unicode] Default: ''

Set the Access-Control-Allow-Origin header

Use '*' to allow any origin to access your server.

Takes precedence over allow_origin_pat.

ServerApp.allow_origin_pat

[Unicode] Default: ''

Use a regular expression for the Access-Control-Allow-Origin header

Requests from an origin matching the expression will get replies with:

Access-Control-Allow-Origin: origin

where *origin* is the origin of the request.

Ignored if allow_origin is set.

ServerApp.allow_password_change

[Bool] Default: True

DEPRECATED in 2.0. Use PasswordIdentityProvider.allow_password_change

ServerApp.allow_remote_access

[Bool] Default: False

Allow requests where the Host header doesn't point to a local server

By default, requests get a 403 forbidden response if the 'Host' header shows that the browser thinks it's on a non-local domain. Setting this option to True disables this check.

This protects against 'DNS rebinding' attacks, where a remote web server serves you a page and then changes its DNS to send later requests to a local IP, bypassing same-origin checks.

Local IP addresses (such as 127.0.0.1 and ::1) are allowed as local, along with hostnames configured in local_hostnames.

ServerApp.allow_root

[Bool] Default: False

Whether to allow the user to run the server as root.

ServerApp.answer_yes

[Bool] Default: False

Answer yes to any prompts.

ServerApp.authenticate_prometheus

[Bool] Default: True

“

Require authentication to access prometheus metrics.

ServerApp.authorizer_class

[Type] Default: 'jupyter_server.auth.authorizer.AllowAllAuthorizer'

The authorizer class to use.

ServerApp.autoreload

[Bool] Default: False

Reload the webapp when changes are made to any Python src files.

ServerApp.base_url

[Unicode] Default: '/'

The base URL for the Jupyter server.

Leading and trailing slashes can be omitted, and will automatically be added.

ServerApp.browser

[Unicode] Default: ''

Specify what command to use to invoke a web

browser when starting the server. If not specified, the default browser will be determined by the *webbrowser* standard library module, which allows setting of the BROWSER environment variable to override it.

ServerApp.certfile

[Unicode] Default: ''

The full path to an SSL/TLS certificate file.

ServerApp.client_ca

[Unicode] Default: ''

The full path to a certificate authority certificate for SSL/TLS client authentication.

ServerApp.config_file

[Unicode] Default: ''

Full path of a config file.

ServerApp.config_file_name

[Unicode] Default: ''

Specify a config file to load.

ServerApp.config_manager_class

[Type] Default: 'jupyter_server.services.config.manager.ConfigManager'

The config manager class to use

ServerApp.contents_manager_class

[Type] Default: 'jupyter_server.services.contents.largefilemanager.AsyncLarge...

The content manager class to use.

ServerApp.cookie_options

[Dict] Default: {}

DEPRECATED. Use IdentityProvider.cookie_options

ServerApp.cookie_secret

[Bytes] Default: b''

The random bytes used to secure cookies.

By default this is a new random number every time you start the server. Set it to a value in a config file to enable logins to persist across server sessions.

Note: Cookie secrets should be kept private, do not share config files with cookie_secret stored in plaintext (you can read the value from a file).

ServerApp.cookie_secret_file

[Unicode] Default: ''

The file where the cookie secret is stored.

ServerApp.custom_display_url

[Unicode] Default: ''

Override URL shown to users.

Replace actual URL, including protocol, address, port and base URL, with the given value when displaying URL to the users. Do not change the actual connection URL. If authentication token is enabled, the token is added to the custom URL automatically.

This option is intended to be used when the URL to display to the user cannot be determined reliably by the Jupyter server (proxified or containerized setups for example).

ServerApp.default_url

[Unicode] Default: '/'

The default URL to redirect to from /

ServerApp.disable_check_xsrf

[Bool] Default: False

Disable cross-site-request-forgery protection

Jupyter server includes protection from cross-site request forgeries, requiring API requests to either:

- originate from pages served by this server (validated with XSRF cookie and token), or
- authenticate with a token

Some anonymous compute resources still desire the ability to run code, completely without authentication. These services can disable all authentication and security checks, with the full knowledge of what that implies.

ServerApp.extra_services

[List] Default: []

handlers that should be loaded at higher priority than the default services

ServerApp.extra_static_paths

[List] Default: []

Extra paths to search for serving static files.

This allows adding javascript/css to be available from the Jupyter server machine, or overriding individual files in the IPython

ServerApp.extra_template_paths

[List] Default: []

Extra paths to search for serving jinja templates.

Can be used to override templates from `jupyter_server.templates`.

ServerApp.file_to_run

[Unicode] Default: ''

Open the named file when the application is launched.

ServerApp.file_url_prefix

[Unicode] Default: 'notebooks'

The URL prefix where files are opened directly.

ServerApp.generate_config

[Bool] Default: False

Generate default config file.

ServerApp.get_secure_cookie_kwargs

[Dict] Default: {}

DEPRECATED. Use IdentityProvider.get_secure_cookie_kwargs

ServerApp.identity_provider_class

[Type] Default: 'jupyter_server.auth.identity.PasswordIdentityProvider'

The identity provider class to use.

ServerApp.iopub_data_rate_limit

[Float] Default: 0.0

DEPRECATED. Use ZMQChannelsWebsocketConnection.iopub_data_rate_limit

ServerApp.iopub_msg_rate_limit

[Float] Default: 0.0

DEPRECATED. Use ZMQChannelsWebsocketConnection.iopub_msg_rate_limit

ServerApp.ip

[Unicode] Default: 'localhost'

The IP address the Jupyter server will listen on.

ServerApp.jinja_environment_options

[Dict] Default: {}

Supply extra arguments that will be passed to Jinja environment.

ServerApp.jinja_template_vars

[Dict] Default: {}

Extra variables to supply to jinja templates when rendering.

ServerApp.jpserver_extensions

[Dict] Default: {}

Dict of Python modules to load as Jupyter server extensions. Entry values can be used to enable and disable the loading of the extensions. The extensions will be loaded in alphabetical order.

ServerApp.kernel_manager_class

[Type] Default: 'jupyter_server.services.kernels.kernelmanager.MappingKernelM...

The kernel manager class to use.

ServerApp.kernel_spec_manager_class

[Type] Default: 'builtins.object'

The kernel spec manager class to use. Should be a subclass of *jupyter_client.kernelspec.KernelSpecManager*.

The Api of KernelSpecManager is provisional and might change without warning between this version of Jupyter and the next stable one.

ServerApp.kernel_websocket_connection_class

[Type] Default: 'jupyter_server.services.kernels.connection.base.BaseKernelWe...

The kernel websocket connection class to use.

ServerApp.kernel_ws_protocol

[Unicode] Default: ''

DEPRECATED. Use `ZMQChannelsWebsocketConnection.kernel_ws_protocol`

ServerApp.keyfile

[Unicode] Default: ''

The full path to a private key file for usage with SSL/TLS.

ServerApp.limit_rate

[Bool] Default: False

DEPRECATED. Use `ZMQChannelsWebsocketConnection.limit_rate`

ServerApp.local_hostnames

[List] Default: ['localhost']

Hostnames to allow as local when `allow_remote_access` is False.

Local IP addresses (such as 127.0.0.1 and ::1) are automatically accepted as local as well.

ServerApp.log_datefmt

[Unicode] Default: '%Y-%m-%d %H:%M:%S'

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

ServerApp.log_format

[Unicode] Default: '%[(name)s]%(highlevel)s %(message)s'

The Logging format template

ServerApp.log_level

[any of 0 `` | `` 10 `` | `` 20 `` | `` 30 `` | `` 40 `` | `` 50 `` | 'DEBUG' | 'INFO' | 'WARN' | 'ERROR' | 'CRITICAL']
Default: 30

Set the log level by value or name.

ServerApp.logging_config

[Dict] Default: {}

Configure additional log handlers.

The default stderr logs handler is configured by the `log_level`, `log_datefmt` and `log_format` settings.

This configuration can be used to configure additional handlers (e.g. to output the log to a file) or for finer control over the default handlers.

If provided this should be a logging configuration dictionary, for more information see: <https://docs.python.org/3/library/logging.config.html#logging-config-dictschema>

This dictionary is merged with the base logging configuration which defines the following:

- A logging formatter intended for interactive use called `console`.
- A logging handler that writes to stderr called `console` which uses the formatter `console`.
- A logger with the name of this application set to `DEBUG` level.

This example adds a new handler that writes to a file:

```
c.Application.logging_config = {
    'handlers': {
        'file': {
            'class': 'logging.FileHandler',
            'level': 'DEBUG',
            'filename': '<path/to/file>',
        }
    }
```

(continues on next page)

(continued from previous page)

```

    },
    'loggers': {
        '<application-name>': {
            'level': 'DEBUG',
            # NOTE: if you don't list the default "console"
            # handler here then it will be disabled
            'handlers': ['console', 'file'],
        },
    },
}

```

ServerApp.login_handler_class

[Type] Default: 'jupyter_server.auth.login.LegacyLoginHandler'

The login handler class to use.

ServerApp.logout_handler_class

[Type] Default: 'jupyter_server.auth.logout.LogoutHandler'

The logout handler class to use.

ServerApp.max_body_size

[Int] Default: 536870912

Sets the maximum allowed size of the client request body, specified in the Content-Length request header field. If the size in a request exceeds the configured value, a malformed HTTP message is returned to the client.

Note: max_body_size is applied even in streaming mode.

ServerApp.max_buffer_size

[Int] Default: 536870912

Gets or sets the maximum amount of memory, in bytes, that is allocated for use by the buffer manager.

ServerApp.min_open_files_limit

[Int] Default: 0

Gets or sets a lower bound on the open file handles process resource limit. This may need to be increased if you run into an OSError: [Errno 24] Too many open files. This is not applicable when running on Windows.

ServerApp.notebook_dir

[Unicode] Default: ''

DEPRECATED, use root_dir.

ServerApp.open_browser

[Bool] Default: False

Whether to open in a browser after starting.

The specific browser used is platform dependent and determined by the python standard library *webbrowser* module, unless it is overridden using the `--browser` (`ServerApp.browser`) configuration option.

ServerApp.password

[Unicode] Default: ''

DEPRECATED in 2.0. Use PasswordIdentityProvider.hashed_password

ServerApp.password_required

[Bool] Default: False

DEPRECATED in 2.0. Use PasswordIdentityProvider.password_required

ServerApp.port

[Int] Default: 0

The port the server will listen on (env: JUPYTER_PORT).

ServerApp.port_retries

[Int] Default: 50

The number of additional ports to try if the specified port is not available (env: JUPYTER_PORT_RETRIES).

ServerApp.preferred_dir

[Unicode] Default: ''

Preferred starting directory to use for notebooks and kernels.

ServerApp.pylab

[Unicode] Default: 'disabled'

DISABLED: use %pylab or %matplotlib in the notebook to enable matplotlib.

ServerApp.quit_button

[Bool] Default: True

If True, display controls to shut down the Jupyter server, such as menu items or buttons.

ServerApp.rate_limit_window

[Float] Default: 0.0

DEPRECATED. Use ZMQChannelsWebsocketConnection.rate_limit_window

ServerApp.reraise_server_extension_failures

[Bool] Default: False

Reraise exceptions encountered loading server extensions?

ServerApp.root_dir

[Unicode] Default: ''

The directory to use for notebooks and kernels.

ServerApp.session_manager_class

[Type] Default: 'builtins.object'

The session manager class to use.

ServerApp.show_config

[Bool] Default: False

Instead of starting the Application, dump configuration to stdout

ServerApp.show_config_json

[Bool] Default: False

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

ServerApp.shutdown_no_activity_timeout

[Int] Default: 0

Shut down the server after N seconds with no kernels running and no activity. This can be used together with culling idle kernels (MappingKernelManager.cull_idle_timeout) to shutdown the Jupyter server when it's not in use. This is not precisely timed: it may shut down up to a minute later. 0 (the default) disables this automatic shutdown.

ServerApp.sock

[Unicode] Default: ''

The UNIX socket the Jupyter server will listen on.

ServerApp.sock_mode

[Unicode] Default: '0600'

The permissions mode for UNIX socket creation (default: 0600).

ServerApp.ssl_options

[Dict] Default: {}

Supply SSL options for the tornado HTTPServer.

See the tornado docs for details.

ServerApp.static_immutable_cache

[List] Default: []

Paths to set up static files as immutable.

This allow setting up the cache control of static files as immutable. It should be used for static file named with a hash for instance.

ServerApp.terminado_settings

[Dict] Default: {}

Supply overrides for terminado. Currently only supports “shell_command”.

ServerApp.terminals_enabled

[Bool] Default: False

Set to False to disable terminals.

This does *not* make the server more secure by itself. Anything the user can in a terminal, they can also do in a notebook.

Terminals may also be automatically disabled if the terminado package is not available.

ServerApp.token

[Unicode] Default: '<DEPRECATED>'

DEPRECATED. Use IdentityProvider.token

ServerApp.tornado_settings

[Dict] Default: {}

Supply overrides for the tornado.web.Application that the Jupyter server uses.

ServerApp.trust_xheaders

[Bool] Default: False

Whether to trust or not X-Scheme/X-Forwarded-Proto and X-Real-Ip/X-Forwarded-For headers sent by the upstream reverse proxy. Necessary if the proxy handles SSL

ServerApp.use_redirect_file

[Bool] Default: True

Disable launching browser by redirect file

For versions of notebook > 5.7.2, a security feature measure was added that prevented the authentication token used to launch the browser from being visible. This feature makes it difficult for other users on a multi-user system from running code in your Jupyter session as you. However, some environments (like Windows Subsystem for Linux (WSL) and Chromebooks), launching a browser using a redirect file can lead the browser failing to load. This is because of the difference in file structures/paths between the runtime and the browser.

Disabling this setting to False will disable this behavior, allowing the browser to launch by using a URL and visible token (as before).

ServerApp.webbrowser_open_new

[Int] Default: 2

Specify where to open the server on startup. This is the

new argument passed to the standard library method *webbrowser.open*. The behaviour is not guaranteed, but depends on browser support. Valid values are:

- 2 opens a new tab,
- 1 opens a new window,
- 0 opens in an existing window.

See the *webbrowser.open* documentation for details.

ServerApp.websocket_compression_options

[Any] Default: None

Set the tornado compression options for websocket connections.

This value will be returned from `WebSocketHandler.get_compression_options()`. None (default) will disable compression. A dict (even an empty one) will enable compression.

See the tornado docs for `WebSocketHandler.get_compression_options` for details.

ServerApp.websocket_url

[Unicode] Default: ''

The base URL for websockets,

if it differs from the HTTP server (hint: it almost certainly doesn't).

Should be in the form of an HTTP origin: `ws[s]://hostname[:port]`

2.3 Running a Jupyter Server with NbClassic

The *Jupyter NbClassic* web application is based on a server-client structure. Jupyter server uses a [two-process kernel architecture](#) based on [ZeroMQ](#), as well as [Tornado](#) for serving HTTP requests.

Note: By default, running `nbclassic` will start a jupyter server which runs locally at `127.0.0.1:8888` and is accessible only from *localhost*. You may access the server from the browser using `http://127.0.0.1:8888`.

This document describes how you can [secure a jupyter server](#) and how to [run it on a public interface](#). For general documentation on Jupyter Server, please visit the section on [Running a public Jupyter Server](#).

Important: **This is not the multi-user server you are looking for.** This document describes how you can run a public server with a single user. This should only be done by someone who wants remote access to their personal machine. Even so, doing this requires a thorough understanding of the set-ups limitations and security implications. If you allow multiple users to access a jupyter server as it is described in this document, their commands may collide, clobber and overwrite each other.

If you want a multi-user server, the official solution is [JupyterHub](#). To use JupyterHub, you need a Unix server (typically Linux) running somewhere that is accessible to your users on a network. This may run over the public internet, but doing so introduces additional [security concerns](#).

2.3.1 Securing a Jupyter server Running with NbClassic

You can protect your jupyter server with a simple single password. As of notebook 5.0 this could be done automatically. To set up a password manually you can configure the `NotebookApp.password` setting in `jupyter_notebook_config.py`.

Prerequisite: A notebook configuration file

Check to see if you have a notebook configuration file, `jupyter_notebook_config.py`. The default location for this file is your Jupyter folder located in your home directory:

- Windows: `C:\Users\USERNAME\.jupyter\jupyter_notebook_config.py`
- OS X: `/Users/USERNAME/.jupyter/jupyter_notebook_config.py`
- Linux: `/home/USERNAME/.jupyter/jupyter_notebook_config.py`

If you don't already have a Jupyter folder, or if your Jupyter folder doesn't contain a notebook configuration file, run the following command:

```
$ jupyter nbclassic --generate-config
```

This command will create the Jupyter folder if necessary, and create notebook configuration file, `jupyter_notebook_config.py`, in this folder.

Automatic Password setup

As of notebook 5.3, the first time you log-in using a token, the notebook server should give you the opportunity to setup a password from the user interface.

You will be presented with a form asking for the current `_token_`, as well as your `_new_ _password_`; enter both and click on **Login** and **setup new password**.

Next time you need to log in you'll be able to use the new password instead of the login token, otherwise follow the procedure to set a password from the command line.

The ability to change the password at first login time may be disabled by integrations by setting the `--NotebookApp.allow_password_change=False`

Starting at notebook version 5.0, you could enter and store a password for your notebook server with a single command. This applies to NbClassic, **jupyter nbclassic password** will prompt you for your password and record the hashed password in your `jupyter_server_config.json`.

```
$ jupyter nbclassic password
Enter password: ****
Verify password: ****
[JupyterPasswordApp] Wrote hashed password to /Users/you/.jupyter/jupyter_server_config.
↪ json
```

This can be used to reset a lost password; or if you believe your credentials have been leaked and desire to change your password. Changing your password will invalidate all logged-in sessions after a server restart.

Preparing a hashed password

You can prepare a hashed password manually, using the function `notebook.auth.security.passwd()`:

```
In [1]: from notebook.auth import passwd
In [2]: passwd()
Enter password:
Verify password:
Out[2]: 'sha1:67c9e60bb8b6:9ffede0825894254b2e042ea597d771089e11aed'
```

Caution: `passwd()` when called with no arguments will prompt you to enter and verify your password such as in the above code snippet. Although the function can also be passed a string as an argument such as `passwd('mypassword')`, please **do not** pass a string as an argument inside an IPython session, as it will be saved in your input history.

Adding hashed password to your notebook configuration file

You can then add the hashed password to your `jupyter_notebook_config.py`. The default location for this file `jupyter_notebook_config.py` is in your Jupyter folder in your home directory, `~/ .jupyter`, e.g.:

```
c.NotebookApp.password = u'sha1:67c9e60bb8b6:9ffede0825894254b2e042ea597d771089e11aed'
```

Automatic password setup will store the hash in `jupyter_server_config.json` while this method stores the hash in `jupyter_notebook_config.py`. The `.json` configuration options take precedence over the `.py` one, thus the manual password may not take effect if the `Json` file has a password set.

Using SSL for encrypted communication

When using a password, it is a good idea to also use SSL with a web certificate, so that your hashed password is not sent unencrypted by your browser.

Important: Web security is rapidly changing and evolving. We provide this document as a convenience to the user, and recommend that the user keep current on changes that may impact security, such as new releases of OpenSSL. The Open Web Application Security Project ([OWASP](https://owasp.org/)) website is a good resource on general security issues and web practices.

You can start `nbclassic` to communicate via a secure protocol mode by setting the `certfile` option to your self-signed certificate, i.e. `mycert.pem`, with the command:

```
$ jupyter nbclassic --certfile=mycert.pem --keyfile mykey.key
```

Tip: A self-signed certificate can be generated with `openssl`. For example, the following command will create a certificate valid for 365 days with both the key and certificate data written to the same file:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mykey.key -out mycert.pem
```

When starting the notebook server, your browser may warn that your self-signed certificate is insecure or unrecognized. If you wish to have a fully compliant self-signed certificate that will not raise warnings, it is possible (but rather involved)

to create one, as explained in detail in this [tutorial](#). Alternatively, you may use [Let's Encrypt](#) to acquire a free SSL certificate and follow the steps in [Using Let's Encrypt](#) to set up a public server.

2.3.2 Running a public Jupyter server with NbClassic

If you want to access your notebook server remotely via a web browser, you can do so by running a public notebook server. For optimal security when running a public jupyter server, you should first secure the server with a password and SSL/HTTPS as described in [Securing a Jupyter server Running with NbClassic](#).

Start by creating a certificate file and a hashed password, as explained in [Securing a Jupyter server Running with NbClassic](#).

If you don't already have one, create a config file for the notebook using the following command line:

```
$ jupyter nbclassic --generate-config
```

In the `~/.` `jupyter` directory, edit the notebook config file, `jupyter_notebook_config.py`. By default, the notebook config file has all fields commented out. The minimum set of configuration options that you should uncomment and edit in `jupyter_notebook_config.py` is the following:

```
# Set options for certfile, ip, password, and toggle off
# browser auto-opening
c.NotebookApp.certfile = u'/absolute/path/to/your/certificate/mycert.pem'
c.NotebookApp.keyfile = u'/absolute/path/to/your/certificate/mykey.key'
# Set ip to '*' to bind on all interfaces (ips) for the public server
c.NotebookApp.ip = '*'
c.NotebookApp.password = u'sha1:bcd259ccf...<your hashed password here>'
c.NotebookApp.open_browser = False

# It is a good idea to set a known, fixed port for server access
c.NotebookApp.port = 9999
```

You can then start the notebook using the `jupyter nbclassic` command.

Using Let's Encrypt

[Let's Encrypt](#) provides free SSL/TLS certificates. You can also set up a public server using a [Let's Encrypt](#) certificate.

[Running a public Jupyter server with NbClassic](#) will be similar when using a [Let's Encrypt](#) certificate with a few configuration changes. Here are the steps:

1. Create a [Let's Encrypt](#) certificate.
2. Use [Preparing a hashed password](#) to create one.
3. If you don't already have config file for the notebook, create one using the following command:

```
$ jupyter nbclassic --generate-config
```

4. In the `~/.` `jupyter` directory, edit the notebook config file, `jupyter_notebook_config.py`. By default, the notebook config file has all fields commented out. The minimum set of configuration options that you should uncomment and edit in `jupyter_notebook_config.py` is the following:

```
# Set options for certfile, ip, password, and toggle off
# browser auto-opening
```

(continues on next page)

(continued from previous page)

```
c.NotebookApp.certfile = u'/absolute/path/to/your/certificate/fullchain.pem'
c.NotebookApp.keyfile = u'/absolute/path/to/your/certificate/privkey.pem'
# Set ip to '*' to bind on all interfaces (ips) for the public server
c.NotebookApp.ip = '*'
c.NotebookApp.password = u'sha1:bcd259ccf...<your hashed password here>'
c.NotebookApp.open_browser = False

# It is a good idea to set a known, fixed port for server access
c.NotebookApp.port = 9999
```

You can then start NbClassic using the `jupyter nbclassic` command.

Important: Use **'https'**. Keep in mind that when you enable SSL support, you must access the notebook server over `https://`, not over plain `http://`. The startup message from the server prints a reminder in the console, but *it is easy to overlook this detail and think the server is for some reason non-responsive*.

When using SSL, always access the notebook server with `'https://'`.

You may now access the public server by pointing your browser to `https://your.host.com:9999` where `your.host.com` is your public server's domain.

Firewall Setup

To function correctly, the firewall on the computer running NbClassic's jupyter server must be configured to allow connections from client machines on the access port `c.NotebookApp.port` set in `jupyter_notebook_config.py` to allow connections to the web interface. The firewall must also allow connections from 127.0.0.1 (localhost) on ports from 49152 to 65535. These ports are used by the server to communicate with the nbclassic kernels. The kernel communication ports are chosen randomly by ZeroMQ, and may require multiple connections per kernel, so a large range of ports must be accessible.

2.3.3 Running the notebook with a customized URL prefix

The notebook dashboard, which is the landing page with an overview of the notebooks in your working directory, is typically found and accessed at the default URL `http://localhost:8888/`.

If you prefer to customize the URL prefix for the notebook dashboard, you can do so through modifying `jupyter_notebook_config.py`. For example, if you prefer that the notebook dashboard be located with a sub-directory that contains other ipython files, e.g. `http://localhost:8888/ipython/`, you can do so with configuration options like the following (see above for instructions about modifying `jupyter_notebook_config.py`):

```
c.NotebookApp.base_url = '/ipython/'
```


2.3.4 Embedding the notebook in another website

Sometimes you may want to embed the notebook somewhere on your website, e.g. in an `IFrame`. To do this, you may need to override the Content-Security-Policy to allow embedding. Assuming your website is at `https://mywebsite.example.com`, you can embed the notebook on your website with the following configuration setting in `jupyter_notebook_config.py`:

```
c.NotebookApp.tornado_settings = {
    'headers': {
        'Content-Security-Policy': "frame-ancestors https://mywebsite.example.com 'self'"
    }
}
```

When embedding the notebook in a website using an `iframe`, consider putting the notebook in single-tab mode. Since the notebook opens some links in new tabs by default, single-tab mode keeps the notebook from opening additional tabs. Adding the following to `~/jupyter/custom/custom.js` will enable single-tab mode:

```
define(['base/js/namespace'], function(Jupyter){
    Jupyter._target = '_self';
});
```

2.3.5 Using a gateway server for kernel management

You are now able to redirect the management of your kernels to a Gateway Server (i.e., [Jupyter Kernel Gateway](#) or [Jupyter Enterprise Gateway](#)) simply by specifying a Gateway url via the following command-line option:

```
$ jupyter nbclassic --gateway-url=http://my-gateway-server:8888
```

the environment:

```
JUPYTER_GATEWAY_URL=http://my-gateway-server:8888
```

or in `jupyter_notebook_config.py`:

```
c.GatewayClient.url = http://my-gateway-server:8888
```

When provided, all kernel specifications will be retrieved from the specified Gateway server and all kernels will be managed by that server. This option enables the ability to target kernel processes against managed clusters while allowing for the notebook's management to remain local to the `NbClassic` server.

2.3.6 Known issues

Below are issues previously reported Notebook server issues and may be applicable to the Jupyter Server package that provides `NbClassic`'s server.

Proxies

When behind a proxy, especially if your system or browser is set to autodetect the proxy, the nbclassic web application might fail to connect to the server's websockets, and present you with a warning at startup. In this case, you need to configure your system not to use the proxy for the server's address.

For example, in Firefox, go to the Preferences panel, Advanced section, Network tab, click 'Settings...', and add the address of the notebook server to the 'No proxy for' field.

Content-Security-Policy (CSP)

Certain [security guidelines](#) recommend that servers use a Content-Security-Policy (CSP) header to prevent cross-site scripting vulnerabilities, specifically limiting to `default-src: https:` when possible. This directive causes two problems with Jupyter. First, it disables execution of inline javascript code, which is used extensively by Jupyter. Second, it limits communication to the https scheme, and prevents WebSockets from working because they communicate via the wss scheme (or ws for insecure communication). Jupyter uses WebSockets for interacting with kernels, so when you visit a server with such a CSP, your browser will block attempts to use wss, which will cause you to see "Connection failed" messages from jupyter notebooks, or simply no response from jupyter terminals. By looking in your browser's javascript console, you can see any error messages that will explain what is failing.

To avoid these problem, you need to add 'unsafe-inline' and `connect-src https: wss:` to your CSP header, at least for pages served by jupyter. (That is, you can leave your CSP unchanged for other parts of your website.) Note that multiple CSP headers are allowed, but successive CSP headers can only restrict the policy; they cannot loosen it. For example, if your server sends both of these headers

```
Content-Security-Policy "default-src https: 'unsafe-inline'" Content-Security-Policy "connect-src https: wss:"
```

the first policy will already eliminate wss connections, so the second has no effect. Therefore, you can't simply add the second header; you have to actually modify your CSP header to look more like this:

```
Content-Security-Policy "default-src https: 'unsafe-inline'; connect-src https: wss:"
```

Docker CMD

Using `jupyter nbclassic` as a [Docker CMD](#) results in kernels repeatedly crashing, likely due to a lack of [PID reaping](#). To avoid this, use the `tini` init as your Dockerfile *ENTRYPOINT*:

```
# Add Tini. Tini operates as a process subreaper for jupyter. This prevents
# kernel crashes.
ENV TINI_VERSION v0.6.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /usr/bin/tini
RUN chmod +x /usr/bin/tini
ENTRYPOINT ["/usr/bin/tini", "--"]

EXPOSE 8888
CMD ["jupyter", "nbclassic", "--port=8888", "--no-browser", "--ip=0.0.0.0"]
```

2.4 Security in the Jupyter NbClassic server

Since access to the Jupyter NbClassic server means access to running arbitrary code, it is important to restrict access to the NbClassic server. For this reason, notebook 4.3 introduced token-based authentication that is **on by default**.

Note: If you enable a password for your NbClassic server, token authentication is not enabled by default, and the behavior of the NbClassic server is unchanged from versions earlier than 4.3.

When token authentication is enabled, the notebook uses a token to authenticate requests. This token can be provided to login to the NbClassic server in three ways:

- in the Authorization header, e.g.:

```
Authorization: token abcdef...
```

- In a URL parameter, e.g.:

```
https://my-notebook/tree/?token=abcdef...
```

- In the password field of the login form that will be shown to you if you are not logged in.

When you start an NbClassic server with token authentication enabled (default), a token is generated to use for authentication. This token is logged to the terminal, so that you can copy/paste the URL into your browser:

```
[I 2022-09-13 09:33:16.916 ServerApp] Jupyter Server 1.18.1 is running at:
[I 2022-09-13 09:33:16.916 ServerApp] http://localhost:8888/tree?
↪ token=c8de56fa4deed24899803e93c227592aef6538f93025fe01
```

If the NbClassic server is going to open your browser automatically (the default, unless `--no-browser` has been passed), an *additional* token is generated for launching the browser. This additional token can be used only once, and is used to set a cookie for your browser once it connects. After your browser has made its first request with this one-time-token, the token is discarded and a cookie is set in your browser.

At any later time, you can see the tokens and URLs for all of your running servers with **jupyter notebook list**:

```
$ jupyter nbclassic list
Currently running servers:
http://localhost:8888/?token=abc... :: /home/you/notebooks
https://0.0.0.0:9999/?token=123... :: /tmp/public
http://localhost:8889/ :: /tmp/has-password
```

For servers with token-authentication enabled, the URL in the above listing will include the token, so you can copy and paste that URL into your browser to login. If a server has no token (e.g. it has a password or has authentication disabled), the URL will not include the token argument. Once you have visited this URL, a cookie will be set in your browser and you won't need to use the token again, unless you switch browsers, clear your cookies, or start a notebook server on a new port.

2.4.1 Alternatives to token authentication

If a generated token doesn't work well for you, you can set a password for your notebook. `jupyter nbclassic password` will prompt you for a password, and store the hashed password in your `jupyter_server_config.json`.

The `jupyter nbclassic password` command is available.

It is possible to disable authentication altogether by setting the token and password to empty strings, but this is **NOT RECOMMENDED**, unless authentication or access restrictions are handled at a different layer in your web application:

```
c.ServerApp.token = ''
c.ServerApp.password = ''
```

2.5 Security in notebook documents

As Jupyter notebooks become more popular for sharing and collaboration, the potential for malicious people to attempt to exploit the notebook for their nefarious purposes increases. IPython 2.0 introduced a security model to prevent execution of untrusted code without explicit user input.

2.5.1 The problem

The whole point of Jupyter is arbitrary code execution. We have no desire to limit what can be done with a notebook, which would negatively impact its utility.

Unlike other programs, a Jupyter notebook document includes output. Unlike other documents, that output exists in a context that can execute code (via Javascript).

The security problem we need to solve is that no code should execute just because a user has **opened** a notebook that **they did not write**. Like any other program, once a user decides to execute code in a notebook, it is considered trusted, and should be allowed to do anything.

2.5.2 Our security model

- Untrusted HTML is always sanitized
- Untrusted Javascript is never executed
- HTML and Javascript in Markdown cells are never trusted
- **Outputs** generated by the user are trusted
- Any other HTML or Javascript (in Markdown cells, output generated by others) is never trusted
- The central question of trust is “Did the current user do this?”

2.5.3 The details of trust

When a notebook is executed and saved, a signature is computed from a digest of the notebook's contents plus a secret key. This is stored in a database, writable only by the current user. By default, this is located at:

```
~/ .local/share/jupyter/nbsignatures.db # Linux
~/Library/Jupyter/nbsignatures.db      # OS X
%APPDATA%/jupyter/nbsignatures.db      # Windows
```

Each signature represents a series of outputs which were produced by code the current user executed, and are therefore trusted.

When you open a notebook, the server computes its signature, and checks if it's in the database. If a match is found, HTML and Javascript output in the notebook will be trusted at load, otherwise it will be untrusted.

Any output generated during an interactive session is trusted.

Updating trust

A notebook's trust is updated when the notebook is saved. If there are any untrusted outputs still in the notebook, the notebook will not be trusted, and no signature will be stored. If all untrusted outputs have been removed (either via `Clear Output` or re-execution), then the notebook will become trusted.

While trust is updated per output, this is only for the duration of a single session. A newly loaded notebook file is either trusted or not in its entirety.

Explicit trust

Sometimes re-executing a notebook to generate trusted output is not an option, either because dependencies are unavailable, or it would take a long time. Users can explicitly trust a notebook in two ways:

- At the command-line, with:

```
jupyter trust /path/to/notebook.ipynb
```

- After loading the untrusted notebook, with `File / Trust Notebook`

These two methods simply load the notebook, compute a new signature, and add that signature to the user's database.

2.5.4 Reporting security issues

If you find a security vulnerability in Jupyter, either a failure of the code to properly implement the model described here, or a failure of the model itself, please report it to security@ipython.org.

If you prefer to encrypt your security reports, you can use [this](#) PGP public key.

2.5.5 Affected use cases

Some use cases that work in Jupyter 1.0 became less convenient in 2.0 as a result of the security changes. We do our best to minimize these annoyances, but security is always at odds with convenience.

Javascript and CSS in Markdown cells

While never officially supported, it had become common practice to put hidden Javascript or CSS styling in Markdown cells, so that they would not be visible on the page. Since Markdown cells are now sanitized (by [Google Caja](#)), all Javascript (including click event handlers, etc.) and CSS will be stripped.

We plan to provide a mechanism for notebook themes, but in the meantime styling the notebook can only be done via either `custom.css` or CSS in HTML output. The latter only have an effect if the notebook is trusted, because otherwise the output will be sanitized just like Markdown.

Collaboration

When collaborating on a notebook, people probably want to see the outputs produced by their colleagues' most recent executions. Since each collaborator's key will differ, this will result in each share starting in an untrusted state. There are three basic approaches to this:

- re-run notebooks when you get them (not always viable)
- explicitly trust notebooks via `jupyter trust` or the notebook menu (annoying, but easy)
- share a notebook signatures database, and use configuration dedicated to the collaboration while working on the project.

To share a signatures database among users, you can configure:

```
c.NotebookNotary.data_dir = "/path/to/signature_dir"
```

to specify a non-default path to the SQLite database (of notebook hashes, essentially). We are aware that SQLite doesn't work well on NFS and we are [working out better ways to do this](#).

2.6 Distributing Jupyter Extensions as Python Packages

2.6.1 Overview

How can the notebook be extended?

The Jupyter Notebook client and server application are both deeply customizable. Their behavior can be extended by creating, respectively:

- nbextension: a notebook extension
 - a single JS file, or directory of JavaScript, Cascading StyleSheets, etc. that contain at minimum a JavaScript module packaged as an [AMD modules](#) that exports a function `load_ipython_extension`
- server extension: an importable Python module
 - that implements `load_jupyter_server_extension`
- bundler extension: an importable Python module with generated File -> Download as / Deploy as menu item trigger

- that implements `bundle`

Why create a Python package for Jupyter extensions?

Since it is rare to have a server extension that does not have any frontend components (an `nbextension`), for convenience and consistency, all these client and server extensions with their assets can be packaged and versioned together as a Python package with a few simple commands, or as of Notebook 5.3, handled automatically by your package manager of choice. This makes installing the package of extensions easier and less error-prone for the user.

2.6.2 Installation of Jupyter Extensions

Install a Python package containing Jupyter Extensions

There are several ways that you may get a Python package containing Jupyter Extensions. Commonly, you will use a package manager for your system:

```
pip install helpful_package
# or
conda install helpful_package
# or
apt-get install helpful_package

# where 'helpful_package' is a Python package containing one or more Jupyter Extensions
```

Automatic installation and Enabling

New in Notebook 5.3

The absolute simplest case requires no user interaction at all! Configured correctly, after installing with their package manager of choice, both server and frontend extensions can be enabled by default in the environment where they were installed, i.e. `--sys-prefix`. See the `setup.py` in the example below.

Enable a Server Extension

The simplest case would be to enable a server extension which has no frontend components.

A `pip` user that wants their configuration stored in their home directory would type the following command:

```
jupyter nbclassic-serverextension enable --py helpful_package
```

Alternatively, a `virtualenv` or `conda` user can pass `--sys-prefix` which keeps their environment isolated and reproducible. For example:

```
# Make sure that your virtualenv or conda environment is activated
[source] activate my-environment

jupyter nbclassic-serverextension enable --py helpful_package --sys-prefix
```

Install the nbextension assets

If a package also has an nbextension with frontend assets that must be available (but not necessarily enabled by default), install these assets with the following command:

```
jupyter nbclassic-extension install --py helpful_package # or --sys-prefix if using
↪ virtualenv or conda
```

Enable nbextension assets

If a package has assets that should be loaded every time a Jupyter app (e.g. lab, notebook, dashboard, terminal) is loaded in the browser, the following command can be used to enable the nbextension:

```
jupyter nbclassic-extension enable --py helpful_package # or --sys-prefix if using
↪ virtualenv or conda
```

2.6.3 Did it work? Check by listing Jupyter Extensions.

After running one or more extension installation steps, you can list what is presently known about nbextensions, server extensions, or bundler extensions. The following commands will list which extensions are available, whether they are enabled, and other extension details:

```
jupyter nbclassic-extension list
jupyter nbclassic-serverextension list
jupyter nbclassic-bundlerextension list
```

2.6.4 Additional resources on creating and distributing packages

Of course, in addition to the files listed, there are number of other files one needs to build a proper package. Here are some good resources: - [The Hitchhiker's Guide to Packaging - Repository Structure and Python](#) by Kenneth Reitz

How you distribute them, too, is important: - [Packaging and Distributing Projects - conda: Building packages](#)

2.6.5 Example - Server extension

Creating a Python package with a server extension

Here is an example of a python module which contains a server extension directly on itself. It has this directory structure:

```
- setup.py
- MANIFEST.in
- my_module/
  - __init__.py
```


Defining the server extension

This example shows that the server extension and its `load_jupyter_server_extension` function are defined in the `__init__.py` file.

`my_module/__init__.py`

```
def _jupyter_server_extension_paths():
    return [{
        "module": "my_module"
    }]

def load_jupyter_server_extension(nbapp):
    nbapp.log.info("my module enabled!")
```

Install and enable the server extension

Which a user can install with:

```
jupyter nbclassic-serverextension enable --py my_module [--sys-prefix]
```

2.6.6 Example - Server extension and nbextension

Creating a Python package with a server extension and nbextension

Here is another server extension, with a front-end module. It assumes this directory structure:

```
- setup.py
- MANIFEST.in
- my_fancy_module/
  - __init__.py
  - static/
    index.js
```

Defining the server extension and nbextension

This example again shows that the server extension and its `load_jupyter_server_extension` function are defined in the `__init__.py` file. This time, there is also a function `_jupyter_nbextension_paths` for the nbextension.

my_fancy_module/__init__.py

```
def _jupyter_server_extension_paths():
    return [{
        "module": "my_fancy_module"
    }]

# Jupyter Extension points
def _jupyter_nbextension_paths():
    return [dict(
        section="notebook",
        # the path is relative to the `my_fancy_module` directory
        src="static",
        # directory in the `nbextension/` namespace
        dest="my_fancy_module",
        # _also_ in the `nbextension/` namespace
        require="my_fancy_module/index")]

def load_jupyter_server_extension(nbapp):
    nbapp.log.info("my module enabled!")
```

Install and enable the server extension and nbextension

The user can install and enable the extensions with the following set of commands:

```
jupyter nbclassic-extension install --py my_fancy_module [--sys-prefix|--user]
jupyter nbclassic-extension enable --py my_fancy_module [--sys-prefix|--system]
jupyter nbclassic-serverextension enable --py my_fancy_module [--sys-prefix|--system]
```

Automatically enabling a server extension and nbextension

New in Notebook 5.3

Server extensions and nbextensions can be installed and enabled without any user intervention or post-install scripts beyond `<package manager> install <extension package name>`

In addition to the `my_fancy_module` file tree, assume:

```
jupyter-config/
├── jupyter_notebook_config.d/
│   └── my_fancy_module.json
└── nbconfig/
    ├── notebook.d/
    │   └── my_fancy_module.json
```

jupyter-config/jupyter_notebook_config.d/my_fancy_module.json

```
{
  "NotebookApp": {
    "nbserver_extensions": {
      "my_fancy_module": true
    }
  }
}
```

jupyter-config/nbconfig/notebook.d/my_fancy_module.json

```
{
  "load_extensions": {
    "my_fancy_module/index": true
  }
}
```

Put all of them in place via:

setup.py

```
import setuptools

setuptools.setup(
    name="MyFancyModule",
    ...
    include_package_data=True,
    data_files=[
        # like `jupyter nbclassic-extension install --sys-prefix`
        ("share/jupyter/nbextensions/my_fancy_module", [
            "my_fancy_module/static/index.js",
        ]),
        # like `jupyter nbclassic-extension enable --sys-prefix`
        ("etc/jupyter/nbconfig/notebook.d", [
            "jupyter-config/nbconfig/notebook.d/my_fancy_module.json"
        ]),
        # like `jupyter nbclassic-serverextension enable --sys-prefix`
        ("etc/jupyter/jupyter_notebook_config.d", [
            "jupyter-config/jupyter_notebook_config.d/my_fancy_module.json"
        ]),
    ],
    ...
    zip_safe=False
)
```

and last, but not least:

MANIFEST.in

```
recursive-include jupyter-config *.json
recursive-include my_fancy_module/static *.js
```

As most package managers will only modify their environment, the eventual configuration will be as if the user had typed:

```
jupyter nbclassic-extension install --py my_fancy_module --sys-prefix
jupyter nbclassic-extension enable --py my_fancy_module --sys-prefix
jupyter nbclassic-serverextension enable --py my_fancy_module --sys-prefix
```

If a user manually disables an extension, that configuration will override the bundled package configuration.

When automagical install fails

Note this can still fail in certain situations with `pip`, requiring manual use of `install` and `enable` commands.

Non-python-specific package managers (e.g. `conda`, `apt`) may choose not to implement the above behavior at the `setup.py` level, having more ways to put data files in various places at build time.

2.6.7 Example - Bundler extension

Creating a Python package with a `bundlerextension`

Here is a bundler extension that adds a *Download as -> Notebook Tarball* (`tar.gz`) option to the notebook *File* menu. It assumes this directory structure:

```
- setup.py
- MANIFEST.in
- my_tarball_bundler/
  - __init__.py
```

Defining the bundler extension

This example shows that the bundler extension and its `bundle` function are defined in the `__init__.py` file.

`my_tarball_bundler/__init__.py`

```
import tarfile
import io
import os
import nbformat

def _jupyter_bundlerextension_paths():
    """Declare bundler extensions provided by this package."""
    return [{
        # unique bundler name
        "name": "tarball_bundler",
```

(continues on next page)

(continued from previous page)

```

    # module containing bundle function
    "module_name": "my_tarball_bundler",
    # human-readable menu item label
    "label" : "Notebook Tarball (tar.gz)",
    # group under 'deploy' or 'download' menu
    "group" : "download",
  }]

```

```

def bundle(handler, model):
    """Create a compressed tarball containing the notebook document.

    Parameters
    -----
    handler : tornado.web.RequestHandler
        Handler that serviced the bundle request
    model : dict
        Notebook model from the configured ContentManager
    """
    notebook_filename = model['name']
    notebook_content = nbformat.writes(model['content']).encode('utf-8')
    notebook_name = os.path.splitext(notebook_filename)[0]
    tar_filename = '{}.tar.gz'.format(notebook_name)

    info = tarfile.TarInfo(notebook_filename)
    info.size = len(notebook_content)

    with io.BytesIO() as tar_buffer:
        with tarfile.open(tar_filename, "w:gz", fileobj=tar_buffer) as tar:
            tar.addfile(info, io.BytesIO(notebook_content))

        # Set headers to trigger browser download
        handler.set_header('Content-Disposition',
                           'attachment; filename="{}".format(tar_filename))
        handler.set_header('Content-Type', 'application/gzip')

        # Return the buffer value as the response
        handler.finish(tar_buffer.getvalue())

```

See *Extending the Notebook* for more documentation about writing nbextensions, server extensions, and bundler extensions.

2.7 Extending the Notebook

Certain subsystems of the notebook server are designed to be extended or overridden by users. These documents explain these systems, and show how to override the notebook's defaults with your own custom behavior.

2.7.1 Contents API

The Jupyter Notebook web application provides a graphical interface for creating, opening, renaming, and deleting files in a virtual filesystem.

The `ContentsManager` class defines an abstract API for translating these interactions into operations on a particular storage medium. The default implementation, `FileContentsManager`, uses the local filesystem of the server for storage and straightforwardly serializes notebooks into JSON. Users can override these behaviors by supplying custom subclasses of `ContentsManager`.

This section describes the interface implemented by `ContentsManager` subclasses. We refer to this interface as the **Contents API**.

Data Model

Filesystem Entities

`ContentsManager` methods represent virtual filesystem entities as dictionaries, which we refer to as **models**.

Models may contain the following entries:

| Key | Type | Info |
|----------------------|-----------------|---|
| name | unicode | Basename of the entity. |
| path | unicode | Full (<i>API-style</i>) path to the entity. |
| type | unicode | The entity type. One of "notebook", "file" or "directory". |
| created | datetime | Creation date of the entity. |
| last_modified | datetime | Last modified date of the entity. |
| content | variable | The "content" of the entity. (<i>See Below</i>) |
| mimetype | unicode or None | The mimetype of content , if any. (<i>See Below</i>) |
| format | unicode or None | The format of content , if any. (<i>See Below</i>) |

Certain model fields vary in structure depending on the `type` field of the model. There are three model types: **notebook**, **file**, and **directory**.

- **notebook models**

- The `format` field is always "json".
- The `mimetype` field is always `None`.
- The `content` field contains a `nbformat.notebooknode.NotebookNode` representing the .ipynb file represented by the model. See the `NBFormat` documentation for a full description.

- **file models**

- The `format` field is either "text" or "base64".
- The `mimetype` field can be any mimetype string, but defaults to `text/plain` for text-format models and `application/octet-stream` for base64-format models. For files with unknown mime types (e.g. unknown file extensions), this field may be `None`.

- The `content` field is always of type `unicode`. For text-format file models, `content` simply contains the file's bytes after decoding as UTF-8. Non-text (base64) files are read as bytes, base64 encoded, and then decoded as UTF-8.

- **directory models**

- The `format` field is always `"json"`.
- The `mimetype` field is always `None`.
- The `content` field contains a list of *content-free* models representing the entities in the directory.

Note: In certain circumstances, we don't need the full content of an entity to complete a Contents API request. In such cases, we omit the `content`, and `format` keys from the model. The default values for the `mimetype` field will might also not be evaluated, in which case it will be set as `None`. This reduced reply most commonly occurs when listing a directory, in which circumstance we represent files within the directory as content-less models to avoid having to recursively traverse and serialize the entire filesystem.

Sample Models

```
# Notebook Model with Content
{
  'content': {
    'metadata': {},
    'nbformat': 4,
    'nbformat_minor': 0,
    'cells': [
      {
        'cell_type': 'markdown',
        'metadata': {},
        'source': 'Some Markdown',
      },
    ],
  },
  'created': datetime(2015, 7, 25, 19, 50, 19, 19865),
  'format': 'json',
  'last_modified': datetime(2015, 7, 25, 19, 50, 19, 19865),
  'mimetype': None,
  'name': 'a.ipynb',
  'path': 'foo/a.ipynb',
  'type': 'notebook',
  'writable': True,
}

# Notebook Model without Content
{
  'content': None,
  'created': datetime.datetime(2015, 7, 25, 20, 17, 33, 271931),
  'format': None,
  'last_modified': datetime.datetime(2015, 7, 25, 20, 17, 33, 271931),
  'mimetype': None,
  'name': 'a.ipynb',
  'path': 'foo/a.ipynb',
  'type': 'notebook',
}
```

(continues on next page)

(continued from previous page)

```
'writable': True
}
```

API Paths

ContentsManager methods represent the locations of filesystem resources as **API-style paths**. Such paths are interpreted as relative to the root directory of the notebook server. For compatibility across systems, the following guarantees are made:

- Paths are always unicode, not bytes.
- Paths are not URL-escaped.
- Paths are always forward-slash (/) delimited, even on Windows.
- Leading and trailing slashes are stripped. For example, /foo/bar/buzz/ becomes foo/bar/buzz.
- The empty string ("") represents the root directory.

Writing a Custom ContentsManager

The default ContentsManager is designed for users running the notebook as an application on a personal computer. It stores notebooks as .ipynb files on the local filesystem, and it maps files and directories in the Notebook UI to files and directories on disk. It is possible to override how notebooks are stored by implementing your own custom subclass of ContentsManager. For example, if you deploy the notebook in a context where you don't trust or don't have access to the filesystem of the notebook server, it's possible to write your own ContentsManager that stores notebooks and files in a database.

Required Methods

A minimal complete implementation of a custom ContentsManager must implement the following methods:

| | |
|---|---|
| ContentsManager.get(path[, content, type, ...]) | Get a file or directory model. |
| ContentsManager.save(model, path) | Save a file or directory model to path. |
| ContentsManager.delete_file(path) | Delete the file or directory at path. |
| ContentsManager.rename_file(old_path, new_path) | Rename a file or directory. |
| ContentsManager.file_exists([path]) | Does a file exist at the given path? |
| ContentsManager.dir_exists(path) | Does a directory exist at the given path? |
| ContentsManager.is_hidden(path) | Is path a hidden directory or file? |

You may be required to specify a Checkpoints object, as the default one, FileCheckpoints, could be incompatible with your custom ContentsManager.

Chunked Saving

The contents API allows for “chunked” saving of files, i.e. saving/transmitting in partial pieces:

- This can only be used when the type of the model is file.
- The model should be as otherwise expected for `save()`, with an added field `chunk`.
- The value of `chunk` should be an integer starting at 1, and incrementing for each subsequent chunk, except for the final chunk, which should be indicated with a value of -1.
- The model returned from using `save()` with `chunk` should be treated as unreliable for all chunks except the final one.
- Any interaction with a file being saved in a chunked manner is unreliable until the final chunk has been saved. This includes directory listings.

Customizing Checkpoints

Customized Checkpoint definitions allows behavior to be altered and extended.

The `Checkpoints` and `GenericCheckpointsMixin` classes (from `notebook.services.contents.checkpoints`) have reusable code and are intended to be used together, but require the following methods to be implemented.

| | |
|--|--|
| <code>Checkpoints.rename_checkpoint(checkpoint_id, ...)</code> | Rename a single checkpoint from <code>old_path</code> to <code>new_path</code> . |
| <code>Checkpoints.list_checkpoints(path)</code> | Return a list of checkpoints for a given file |
| <code>Checkpoints.delete_checkpoint(checkpoint_id, ...)</code> | delete a checkpoint for a file |
| <code>GenericCheckpointsMixin.create_file_checkpoint(...)</code> | Create a checkpoint of the current state of a file |
| <code>GenericCheckpointsMixin.create_notebook_checkpoint(nb, ...)</code> | Create a checkpoint of the current state of a file |
| <code>GenericCheckpointsMixin.get_file_checkpoint(...)</code> | Get the content of a checkpoint for a non-notebook file. |
| <code>GenericCheckpointsMixin.get_notebook_checkpoint(...)</code> | Get the content of a checkpoint for a notebook. |

No-op example

Here is an example of a no-op checkpoints object - note the mixin comes first. The docstrings indicate what each method should do or return for a more complete implementation.

```
class NoOpCheckpoints(GenericCheckpointsMixin, Checkpoints):
    """requires the following methods:"""
    def create_file_checkpoint(self, content, format, path):
        """ -> checkpoint model"""
    def create_notebook_checkpoint(self, nb, path):
        """ -> checkpoint model"""
    def get_file_checkpoint(self, checkpoint_id, path):
        """ -> {'type': 'file', 'content': <str>, 'format': {'text', 'base64'}}"""
    def get_notebook_checkpoint(self, checkpoint_id, path):
        """ -> {'type': 'notebook', 'content': <output of nbformat.read>}"""
```

(continues on next page)

(continued from previous page)

```
def delete_checkpoint(self, checkpoint_id, path):
    """deletes a checkpoint for a file"""
def list_checkpoints(self, path):
    """returns a list of checkpoint models for a given file,
    default just does one per file
    """
    return []
def rename_checkpoint(self, checkpoint_id, old_path, new_path):
    """renames checkpoint from old path to new path"""
```

See `GenericFileCheckpoints` in `notebook.services.contents.filecheckpoints` for a more complete example.

Testing

`notebook.services.contents.tests` includes several test suites written against the abstract `Contents API`. This means that an excellent way to test a new `ContentsManager` subclass is to subclass our tests to make them use your `ContentsManager`.

Note: `PGContents` is an example of a complete implementation of a custom `ContentsManager`. It stores notebooks and files in `PostgreSQL` and encodes directories as SQL relations. `PGContents` also provides an example of how to re-use the notebook's tests.

2.7.2 File save hooks

You can configure functions that are run whenever a file is saved. There are two hooks available:

- `ContentsManager.pre_save_hook` runs on the API path and model with content. This can be used for things like stripping output that people don't like adding to VCS noise.
- `FileContentsManager.post_save_hook` runs on the filesystem path and model without content. This could be used to commit changes after every save, for instance.

They are both called with keyword arguments:

```
pre_save_hook(model=model, path=path, contents_manager=cm)
post_save_hook(model=model, os_path=os_path, contents_manager=cm)
```

Examples

These can both be added to `jupyter_notebook_config.py`.

A pre-save hook for stripping output:

```
def scrub_output_pre_save(model, **kwargs):
    """scrub output before saving notebooks"""
    # only run on notebooks
    if model['type'] != 'notebook':
        return
    # only run on nbformat v4
```

(continues on next page)

(continued from previous page)

```

if model['content']['nbformat'] != 4:
    return

for cell in model['content']['cells']:
    if cell['cell_type'] != 'code':
        continue
    cell['outputs'] = []
    cell['execution_count'] = None

```

```
c.FileContentsManager.pre_save_hook = scrub_output_pre_save
```

A post-save hook to make a script equivalent whenever the notebook is saved (replacing the `--script` option in older versions of the notebook):

```

import io
import os
from notebook.utils import to_api_path

_script_exporter = None

def script_post_save(model, os_path, contents_manager, **kwargs):
    """convert notebooks to Python script after save with nbconvert

    replaces `jupyter notebook --script`
    """
    from nbconvert.exporters.script import ScriptExporter

    if model['type'] != 'notebook':
        return

    global _script_exporter

    if _script_exporter is None:
        _script_exporter = ScriptExporter(parent=contents_manager)

    log = contents_manager.log

    base, ext = os.path.splitext(os_path)
    script, resources = _script_exporter.from_filename(os_path)
    script_fname = base + resources.get('output_extension', '.txt')
    log.info("Saving script %s", to_api_path(script_fname, contents_manager.root_dir))

    with io.open(script_fname, 'w', encoding='utf-8') as f:
        f.write(script)

```

```
c.FileContentsManager.post_save_hook = script_post_save
```

This could be a simple call to `jupyter nbconvert --to script`, but spawning the subprocess every time is quite slow.

2.7.3 Custom request handlers

The notebook webserver can be interacted with using a well defined [RESTful API](#). You can define custom RESTful API handlers in addition to the ones provided by the notebook. As described below, to define a custom handler you need to first write a notebook server extension. Then, in the extension, you can register the custom handler.

Writing a notebook server extension

The notebook webserver is written in Python, hence your server extension should be written in Python too. Server extensions, like IPython extensions, are Python modules that define a specially named load function, `load_jupyter_server_extension`. This function is called when the extension is loaded.

```
def load_jupyter_server_extension(nb_server_app):
    """
    Called when the extension is loaded.

    Args:
        nb_server_app (NotebookWebApplication): handle to the Notebook webserver_
    instance.
    """
    pass
```

To get the notebook server to load your custom extension, you'll need to add it to the list of extensions to be loaded. You can do this using the config system. `NotebookApp.nbserver_extensions` is a config variable which is a dictionary of strings, each a Python module to be imported, mapping to `True` to enable or `False` to disable each extension. Because this variable is notebook config, you can set it two different ways, using config files or via the command line.

For example, to get your extension to load via the command line add a double dash before the variable name, and put the Python dictionary in double quotes. If your package is “mypackage” and module is “mymodule”, this would look like `jupyter notebook --NotebookApp.nbserver_extensions="{ 'mypackage.mymodule': True }"`. Basically the string should be Python importable.

Alternatively, you can have your extension loaded regardless of the command line args by setting the variable in the Jupyter config file. The default location of the Jupyter config file is `~/.jupyter/jupyter_notebook_config.py` (see [Configuration Overview](#)). Inside the config file, you can use Python to set the variable. For example, the following config does the same as the previous command line example.

```
c = get_config()
c.NotebookApp.nbserver_extensions = {
    'mypackage.mymodule': True,
}
```

Before continuing, it's a good idea to verify that your extension is being loaded. Use a print statement to print something unique. Launch the notebook server and you should see your statement printed to the console.

Registering custom handlers

Once you've defined a server extension, you can register custom handlers because you have a handle to the Notebook server app instance (`nb_server_app` above). However, you first need to define your custom handler. To declare a custom handler, inherit from `notebook.base.handlers.IPythonHandler`. The example below[1] is a Hello World handler:

```
from notebook.base.handlers import IPythonHandler

class HelloWorldHandler(IPythonHandler):
    def get(self):
        self.finish('Hello, world!')
```

The Jupyter Notebook server use [Tornado](#) as its web framework. For more information on how to implement request handlers, refer to the [Tornado documentation on the matter](#).

After defining the handler, you need to register the handler with the Notebook server. See the following example:

```
web_app = nb_server_app.web_app
host_pattern = '.*$'
route_pattern = url_path_join(web_app.settings['base_url'], '/hello')
web_app.add_handlers(host_pattern, [(route_pattern, HelloWorldHandler)])
```

Putting this together with the extension code, the example looks like the following:

```
from notebook.utils import url_path_join
from notebook.base.handlers import IPythonHandler

class HelloWorldHandler(IPythonHandler):
    def get(self):
        self.finish('Hello, world!')

def load_jupyter_server_extension(nb_server_app):
    """
    Called when the extension is loaded.

    Args:
        nb_server_app (NotebookWebApplication): handle to the Notebook webserver_
        ↪instance.
    """
    web_app = nb_server_app.web_app
    host_pattern = '.*$'
    route_pattern = url_path_join(web_app.settings['base_url'], '/hello')
    web_app.add_handlers(host_pattern, [(route_pattern, HelloWorldHandler)])
```

Extra Parameters and authentication

Here is a quick rundown of what you need to know to pass extra parameters to the handler and enable authentication:

- extra arguments to the `__init__` constructor are given in a dictionary after the handler class in `add_handlers`:

```
class HelloWorldHandler(IPythonHandler):

    def __init__(self, *args, **kwargs):
        self.extra = kwargs.pop('extra')
        ...

def load_jupyter_server_extension(nb_server_app):

    ...

    web_app.add_handlers(host_pattern,
        [
            (route_pattern, HelloWorldHandler, {"extra": nb_server_app.extra})
        ])

```

All handler methods that require authentication MUST be decorated with `@tornado.web.authenticated`:

```
from tornado import web

class HelloWorldHandler(IPythonHandler):

    ...

    @web.authenticated
    def get(self, *args, **kwargs):
        ...

    @web.authenticated
    def post(self, *args, **kwargs):
        ...

```

References:

1. [Peter Parente's Mindtrove](#)

2.7.4 Custom front-end extensions

This describes the basic steps to write a JavaScript extension for the Jupyter notebook front-end. This allows you to customize the behaviour of the various pages like the dashboard, the notebook, or the text editor.

The structure of a front-end extension

Note: The notebook front-end and Javascript API are not stable, and are subject to a lot of changes. Any extension written for the current notebook is almost guaranteed to break in the next release.

A front-end extension is a JavaScript file that defines an [AMD module](#) which exposes at least a function called `load_ipython_extension`, which takes no arguments. We will not get into the details of what each of these terms consists of yet, but here is the minimal code needed for a working extension:

```
// file my_extension/main.js

define(function(){

    function load_ipython_extension(){
        console.info('this is my first extension');
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

Note: Although for historical reasons the function is called `load_ipython_extension`, it does apply to the Jupyter notebook in general, and will work regardless of the kernel in use.

If you are familiar with JavaScript, you can use this template to require any Jupyter module and modify its configuration, or do anything else in client-side Javascript. Your extension will be loaded at the right time during the notebook page initialisation for you to set up a listener for the various events that the page can trigger.

You might want access to the current instances of the various Jupyter notebook components on the page, as opposed to the classes defined in the modules. The current instances are exposed by a module named `base/js/namespace`. If you plan on accessing instances on the page, you should **require** this module rather than accessing the global variable `Jupyter`, which will be removed in future. The following example demonstrates how to access the current notebook instance:

```
// file my_extension/main.js

define([
    'base/js/namespace'
], function(
    Jupyter
) {
    function load_ipython_extension() {
        console.log(
            'This is the current notebook application instance:',
```

(continues on next page)

(continued from previous page)

```
        Jupyter.notebook
    );
}

return {
    load_ipython_extension: load_ipython_extension
};
});
```

Modifying key bindings

One of the abilities of extensions is to modify key bindings, although once again this is an API which is not guaranteed to be stable. However, custom key bindings are frequently requested, and are helpful to increase accessibility, so in the following we show how to access them.

Here is an example of an extension that will unbind the shortcut `0,0` in command mode, which normally restarts the kernel, and bind `0,0,0` in its place:

```
// file my_extension/main.js

define([
    'base/js/namespace'
], function(
    Jupyter
) {

    function load_ipython_extension() {
        Jupyter.keyboard_manager.command_shortcuts.remove_shortcut('0,0');
        Jupyter.keyboard_manager.command_shortcuts.add_shortcut('0,0,0', 'jupyter-
↩notebook:restart-kernel');
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

Note: The standard keybindings might not work correctly on non-US keyboards. Unfortunately, this is a limitation of browser implementations and the status of keyboard event handling on the web in general. We appreciate your feedback if you have issues binding keys, or have any ideas to help improve the situation.

You can see that I have used the **action name** `jupyter-notebook:restart-kernel` to bind the new shortcut. There is no API yet to access the list of all available *actions*, though the following in the JavaScript console of your browser on a notebook page should give you an idea of what is available:

```
Object.keys(require('base/js/namespace').actions._actions);
```

In this example, we changed a keyboard shortcut in **command mode**; you can also customize keyboard shortcuts in **edit mode**. However, most of the keyboard shortcuts in edit mode are handled by CodeMirror, which supports custom key bindings via a completely different API.

Defining and registering your own actions

As part of your front-end extension, you may wish to define actions, which can be attached to toolbar buttons, or called from the command palette. Here is an example of an extension that defines an (not very useful!) action to show an alert, and adds a toolbar button using the full action name:

```
// file my_extension/main.js

define([
  'base/js/namespace'
], function(
  Jupyter
) {
  function load_ipython_extension() {

    var handler = function () {
      alert('this is an alert from my_extension!');
    };

    var action = {
      icon: 'fa-comment-o', // a font-awesome class used on buttons, etc
      help : 'Show an alert',
      help_index : 'zz',
      handler : handler
    };
    var prefix = 'my_extension';
    var action_name = 'show-alert';

    var full_action_name = Jupyter.actions.register(action, action_name, prefix); //
    ↪returns 'my_extension:show-alert'
    Jupyter.toolbar.add_buttons_group([full_action_name]);
  }

  return {
    load_ipython_extension: load_ipython_extension
  };
});
```

Every action needs a name, which, when joined with its prefix to make the full action name, should be unique. Built-in actions, like the `jupyter-notebook:restart-kernel` we bound in the earlier *Modifying key bindings* example, use the prefix `jupyter-notebook`. For actions defined in an extension, it makes sense to use the extension name as the prefix. For the action name, the following guidelines should be considered:

- First pick a noun and a verb for the action. For example, if the action is “restart kernel,” the verb is “restart” and the noun is “kernel”.
- Omit terms like “selected” and “active” by default, so “delete-cell”, rather than “delete-selected-cell”. Only provide a scope like “-all-” if it is other than the default “selected” or “active” scope.
- If an action has a secondary action, separate the secondary action with “-and-”, so “restart-kernel-and-clear-output”.
- Use above/below or previous/next to indicate spatial and sequential relationships.
- Don’t ever use before/after as they have a temporal connotation that is confusing when used in a spatial context.
- For dialogs, use a verb that indicates what the dialog will accomplish, such as “confirm-restart-kernel”.

Installing and enabling extensions

You can install your nbextension with the command:

```
jupyter nbclassic-extension install path/to/my_extension/ [--user|--sys-prefix]
```

The default installation is system-wide. You can use `--user` to do a per-user installation, or `--sys-prefix` to install to Python's prefix (e.g. in a virtual or conda environment). Where `my_extension` is the directory containing the Javascript files. This will copy it to a Jupyter data directory (the exact location is platform dependent - see [jupyter_path](#)).

For development, you can use the `--symlink` flag to symlink your extension rather than copying it, so there's no need to reinstall after changes.

To use your extension, you'll also need to **enable** it, which tells the notebook interface to load it. You can do that with another command:

```
jupyter nbclassic-extension enable my_extension/main [--sys-prefix][--section='common']
```

The argument refers to the Javascript module containing your `load_ipython_extension` function, which is `my_extension/main.js` in this example. The `--section='common'` argument will affect all pages, by default it will be loaded on the notebook view only. There is a corresponding `disable` command to stop using an extension without uninstalling it.

Changed in version 4.2: Added `--sys-prefix` argument

Kernel Specific extensions

Warning: This feature serves as a stopgap for kernel developers who need specific JavaScript injected onto the page. The availability and API are subject to change at anytime.

It is possible to load some JavaScript on the page on a per kernel basis. Be aware that doing so will make the browser page reload without warning as soon as the user switches the kernel without notice.

If you, a kernel developer, need a particular piece of JavaScript to be loaded on a “per kernel” basis, such as:

- if you are developing a CodeMirror mode for your language
- if you need to enable some specific debugging options

your `kernelspecs` are allowed to contain a `kernel.js` file that defines an AMD module. The AMD module should define an `onload` function that will be called when the kernelspec loads, such as:

- when you load a notebook that uses your kernelspec
- change the active kernelspec of a notebook to your kernelspec.

Note that adding a `kernel.js` to your kernelspec will add an unexpected side effect to changing a kernel in the notebook. As it is impossible to “unload” JavaScript, any attempt to change the kernelspec again will save the current notebook and reload the page without confirmations.

Here is an example of `kernel.js`:

```
define(function(){
  return {onload: function(){
    console.info('Kernel specific javascript loaded');

    // do more things here, like define a codemirror mode
```

(continues on next page)

(continued from previous page)

```
}}
});
```

2.7.5 Customize keymaps

Note: Declarative Custom Keymaps is a provisional feature with unstable API which is not guaranteed to be kept in future versions of the notebook, and can be removed or changed without warnings.

The notebook shortcuts that are defined by jupyter both in edit mode and command mode are configurable in the frontend configuration file `~/.jupyter/nbconfig/notebook.json`. The modification of keyboard shortcuts suffers from several limitations, mainly that your Browser and OS might prevent certain shortcuts from working correctly. If this is the case, there is unfortunately not much that can be done. The second issue can arise with keyboards that have a layout different than US English. Again, even if we are aware of the issue, there is not much that can be done.

Shortcuts are also limited by the underlying library that handles code and text editing: CodeMirror. If some keyboard shortcuts are conflicting, the method described below might not work to create new keyboard shortcuts, especially in the `edit` mode of the notebook.

The 4 sections of interest in `~/.jupyter/nbconfig/notebook.json` are the following:

- `keys.command.unbind`
- `keys.edit.unbind`
- `keys.command.bind`
- `keys.edit.bind`

The first two sections describe which default keyboard shortcuts not to register at notebook startup time. These are mostly useful if you need to `unbind` a default keyboard shortcut before binding it to a new `command`.

The first two sections apply respectively to the `command` and `edit` mode of the notebook. They take a list of shortcuts to `unbind`.

For example, to unbind the shortcut to split a cell at the position of the cursor (`Ctrl-Shift-Minus`) use the following:

```
// file ~/.jupyter/nbconfig/notebook.json

{
  "keys": {
    "edit": {
      "unbind": [
        "Ctrl-Shift-Minus"
      ]
    },
  },
}
```

The last two sections describe which new keyboard shortcuts to register at notebook startup time and which actions they trigger.

The last two sections apply respectively to the `command` and `edit` mode of the notebook. They take a dictionary with shortcuts as `keys` and `commands` name as value.

For example, to bind the shortcut G, G, G (Press G three time in a row) in command mode to the command that restarts the kernel and runs all cells, use the following:

```
// file ~/.jupyter/nbconfig/notebook.json

{
  "keys": {
    "command": {
      "bind": {
        "G,G,G": "jupyter-notebook:restart-kernel-and-run-all-cells"
      }
    }
  },
}
```

The name of the available commands can be find by hovering over the right end of a row in the command palette.

2.7.6 Custom bundler extensions

The notebook server supports the writing of *bundler extensions* that transform, package, and download/deploy notebook files. As a developer, you need only write a single Python function to implement a bundler. The notebook server automatically generates a *File -> Download as* or *File -> Deploy as* menu item in the notebook front-end to trigger your bundler.

Here are some examples of what you can implement using bundler extensions:

- Convert a notebook file to a HTML document and publish it as a post on a blog site
- Create a snapshot of the current notebook environment and bundle that definition plus notebook into a zip download
- Deploy a notebook as a standalone, interactive [dashboard](#)

To implement a bundler extension, you must do all of the following:

- Declare bundler extension metadata in your Python package
- Write a *bundle* function that responds to bundle requests
- Instruct your users on how to enable/disable your bundler extension

The following sections describe these steps in detail.

Declaring bundler metadata

You must provide information about the bundler extension(s) your package provides by implementing a `_jupyter_bundlerextensions_paths` function. This function can reside anywhere in your package so long as it can be imported when enabling the bundler extension. (See *Enabling/disabling bundler extensions*.)

```
# in mypackage.hello_bundler

def _jupyter_bundlerextension_paths():
    """Example "hello world" bundler extension"""
    return [{
        'name': 'hello_bundler',          # unique bundler name
        'label': 'Hello Bundler',        # human-readable menu item label
```

(continues on next page)

(continued from previous page)

```

    'module_name': 'mypackage.hello_bundler',    # module containing bundle()
    'group': 'deploy'                            # group under 'deploy' or 'download'
    ↪ menu
  }]

```

Note that the return value is a list. By returning multiple dictionaries in the list, you allow users to enable/disable sets of bundlers all at once.

Writing the *bundle* function

At runtime, a menu item with the given label appears either in the *File -> Deploy as* or *File -> Download as* menu depending on the *group* value in your metadata. When a user clicks the menu item, a new browser tab opens and notebook server invokes a *bundle* function in the *module_name* specified in the metadata.

You must implement a *bundle* function that matches the signature of the following example:

```

# in mypackage.hello_bundler

def bundle(handler, model):
    """Transform, convert, bundle, etc. the notebook referenced by the given
    model.

    Then issue a Tornado web response using the `handler` to redirect
    the user's browser, download a file, show a HTML page, etc. This function
    must finish the handler response before returning either explicitly or by
    raising an exception.

    Parameters
    -----
    handler : tornado.web.RequestHandler
        Handler that serviced the bundle request
    model : dict
        Notebook model from the configured ContentManager
    """
    handler.finish('I bundled {}'.format(model['path']))

```

Your *bundle* function is free to do whatever it wants with the request and respond in any manner. For example, it may read additional query parameters from the request, issue a redirect to another site, run a local process (e.g., *nbconvert*), make a HTTP request to another service, etc.

The caller of the *bundle* function is `@tornado.gen.coroutine` decorated and wraps its call with `torando.gen.maybe_future`. This behavior means you may handle the web request synchronously, as in the example above, or asynchronously using `@tornado.gen.coroutine` and `yield`, as in the example below.

```

from tornado import gen

@gen.coroutine
def bundle(handler, model):
    # simulate a long running IO op (e.g., deploying to a remote host)
    yield gen.sleep(10)

    # now respond
    handler.finish('I spent 10 seconds bundling {}'.format(model['path']))

```

You should prefer the second, asynchronous approach when your bundle operation is long-running and would otherwise block the notebook server main loop if handled synchronously.

For more details about the data flow from menu item click to bundle function invocation, see [Bundler invocation details](#).

Enabling/disabling bundler extensions

The notebook server includes a command line interface (CLI) for enabling and disabling bundler extensions.

You should document the basic commands for enabling and disabling your bundler. One possible command for enabling the *hello_bundler* example is the following:

```
jupyter nbclassic-bundlerextension enable --py mypackage.hello_bundler --sys-prefix
```

The above updates the notebook configuration file in the current conda/virtualenv environment (*--sys-prefix*) with the metadata returned by the *mypackage.hello_bundler._jupyter_bundlerextension_paths* function.

The corresponding command to later disable the bundler extension is the following:

```
jupyter nbclassic-bundlerextension disable --py mypackage.hello_bundler --sys-prefix
```

For more help using the *bundlerextension* subcommand, run the following.

```
jupyter nbclassic-bundlerextension --help
```

The output describes options for listing enabled bundlers, configuring bundlers for single users, configuring bundlers system-wide, etc.

Example: IPython Notebook bundle (.zip)

The *hello_bundler* example in this documentation is simplistic in the name of brevity. For more meaningful examples, see *notebook/bundler/zip_bundler.py* and *notebook/bundler/tarball_bundler.py*. You can enable them to try them like so:

```
jupyter nbclassic-bundlerextension enable --py notebook.bundler.zip_bundler --sys-prefix
jupyter nbclassic-bundlerextension enable --py notebook.bundler.tarball_bundler --sys-
↪ prefix
```

Bundler invocation details

Support for bundler extensions comes from Python modules in *notebook/bundler* and JavaScript in *nbclassic/static/notebook/js/menubar.js*. The flow of data between the various components proceeds roughly as follows:

1. User opens a notebook document
2. Notebook front-end JavaScript loads notebook configuration
3. Bundler front-end JS creates menu items for all bundler extensions in the config
4. User clicks a bundler menu item
5. JS click handler opens a new browser window/tab to `<notebook base_url>/bundle/<path/to/notebook>?bundler=<name>` (i.e., a HTTP GET request)
6. Bundle handler validates the notebook path and bundler *name*
7. Bundle handler delegates the request to the *bundle* function in the bundler's *module_name*

8. *bundle* function finishes the HTTP request

Find out more about configuring NbClassic.

CONTRIBUTE

3.1 Contributing to Jupyter NbClassic

If you're reading this section, you're probably interested in contributing to Jupyter. Welcome and thanks for your interest in contributing!

Please take a look at the Contributor documentation, familiarize yourself with using the Jupyter NbClassic, and introduce yourself on the mailing list and share what area of the project you are interested in working on.

3.1.1 General Guidelines

For general documentation about contributing to Jupyter projects, see the [Project Jupyter Contributor Documentation](#).

3.1.2 Setting Up a Development Environment

Installing Node.js and npm

Building the NbClassic from its GitHub source code requires some tools to create and minify JavaScript components and the CSS, specifically Node.js and Node's package manager, `npm`. It should be node version 6.0.

If you use `conda`, you can get them with:

```
conda install -c conda-forge nodejs
```

If you use [Homebrew](#) on Mac OS X:

```
brew install node
```

Installation on Linux may vary, but be aware that the `nodejs` or `npm` packages included in the system package repository may be too old to work properly.

You can also use the installer from the [Node.js website](#).

Installing the Jupyter NbClassic

Once you have installed the dependencies mentioned above, use the following steps:

```
pip install --upgrade setuptools pip
git clone https://github.com/jupyter/nbclassic
cd nbclassic
pip install -e .
```

If you are using a system-wide Python installation and you only want to install NbClassic for you, you can add `--user` to the install commands.

Once you have done this, you can launch the main branch of Jupyter NbClassic from any directory in your system with:

```
jupyter nbclassic
```

Verification

While running NbClassic, select one of your notebook files (the file will have the extension `.ipynb`). In the top tab you will click on “Help” and then click on “About”. In the pop window you will see information about the version of Jupyter that you are running. You will see “The version of the notebook server is:”. If you are working in development mode, you will see that your version of Jupyter NbClassic will include the word “dev”. If it does not include the word “dev”, you are currently not working in development mode and should follow the steps below to uninstall and reinstall Jupyter.

Troubleshooting the Installation

If you do not see that your Jupyter NbClassic is running on dev mode, it’s possible that you are running other instances of Jupyter NbClassic. You can try the following steps:

1. Uninstall all instances of the NbClassic package. These include any installations you made using pip or conda.
2. Run `python3 -m pip install -e .` in the NbClassic repository to install NbClassic from there.
3. Run `npm run build` to make sure the Javascript and CSS are updated and compiled.
4. Launch with `python3 -m nbclassic --port 8989`, and check that the browser is pointing to `localhost:8989` (rather than the default 8888). You don’t necessarily have to launch with port 8989, as long as you use a port that is neither the default nor in use, then it should be fine.
5. Verify the installation with the steps in the previous section.

Rebuilding JavaScript and CSS

There is a build step for the JavaScript and CSS in the nbclassic. To make sure that you are working with up-to-date code, you will need to run this command whenever there are changes to JavaScript or LESS sources:

```
npm run build
```

IMPORTANT: Don’t forget to run `npm run build` after switching branches. When switching between branches of different versions (e.g. `4.x` and `main`), run `pip install -e ..`. If you have tried the above and still find that NbClassic is not reflecting the current source code, try cleaning the repo with `git clean -xdf` and reinstalling with `pip install -e ..`.

Development Tip

When doing development, you can use this command to automatically rebuild JavaScript and LESS sources as they are modified:

```
npm run build:watch
```

Git Hooks

If you want to automatically update dependencies and recompile JavaScript and CSS after checking out a new commit, you can install post-checkout and post-merge hooks which will do it for you:

```
git-hooks/install-hooks.sh
```

See `git-hooks/README.md` for more details.

3.1.3 Running Tests

Python Tests

Install dependencies:

```
pip install -e '.[test]'
```

To run the Python tests, use:

```
pytest
```

For the end to end Pytest-Playwright tests you will need to install the browser binaries using:

```
playwright install
```

Then you can run the end to end tests using:

```
pytest -sv nbclassic/tests/end_to_end
```

If you want coverage statistics as well, you can run:

```
py.test --cov nbclassic -v --pyargs nbclassic
```

JavaScript Tests

To run the JavaScript tests, you will need to have PhantomJS and CasperJS installed:

```
npm install -g casperjs phantomjs-prebuilt
```

Then, to run the JavaScript tests:

```
python -m nbclassic.jstest [group]
```

where `[group]` is an optional argument that is a path relative to `nbclassic/tests/`. For example, to run all tests in `nbclassic/tests/notebook`:

```
python -m nbclassic.jstest notebook
```

or to run just `nbclassic/tests/notebook/deletecell.js`:

```
python -m nbclassic.jstest notebook/deletecell.js
```

3.1.4 Building the Documentation

To build the documentation you'll need [Sphinx](#), [pandoc](#) and a few other packages.

To install (and activate) a conda environment named `nbclassic_docs` containing all the necessary packages (except `pandoc`), use:

```
conda create -n nbclassic_docs pip
conda activate nbclassic_docs # Linux and OS X
activate nbclassic_docs      # Windows
pip install .[docs]
```

If you want to install the necessary packages with `pip`, use the following instead:

```
pip install .[docs]
```

Once you have installed the required packages, you can build the docs with:

```
cd docs
make html
```

After that, the generated HTML files will be available at `build/html/index.html`. You may view the docs in your browser.

You can automatically check if all hyperlinks are still valid:

```
make linkcheck
```

Windows users can find `make.bat` in the docs folder.

You should also have a look at the [Project Jupyter Documentation Guide](#).

3.2 The Development of NbClassic

Here you will find information about some especially noteworthy updates made in NbClassic, issues that have been reported and common questions.

3.2.1 Noteworthy Updates in NbClassic

- **Entrypoints in NbClassic**
 - In NbClassic entrypoints have been renamed ([Rename duplicate entrypoints #138](#)) to:
 - * `jupyter-nbclassic-extension`
 - * `jupyter nbclassic-serverextension`
 - * `jupyter-nbclassic-bundlerextension`
 - The decision to rename these entrypoints came about after some deliberation and consideration for user experience. When considering the confusion that having a more implicit handling of the entrypoints might pose, the consensus was that renaming the entrypoints would allow for more observability and it would help highlight some of the changes that are happening in the Jupyter ecosystem.
- **Providing backwards compatibility with the `jupyter_notebook_config.py` file**
 - With the goal of allowing NbClassic to be installed along with Notebook 7, the release of NbClassic 0.4 included changing the project name from *notebook* to *nbclassic*. In changing the ‘name’ attribute to be *nbclassic*, the traitlet behavior changed and resulted in the configuration file which was previously named `jupyter_notebook_config`, to be named `jupyter_nbclassic_config`. However, this was updated to manually set the file name to `jupyter_notebook_config`. With this, the configuration file is picked up whether Notebook or NbClassic are installed.
- **Endpoints in NbClassic**
 - NbClassic handlers have been updated to account for Notebook 7 being installed ([Handlers under nbclassic if notebook 7 is found](#)). If so, the resources from nbclassic will be served under the `/nbclassic/` URL subpath, so as to not interfere with those resources being served by Jupyter Notebook.
 - Redirecting from `/tree` to `/nbclassic/tree` if both Notebook 7 and NbClassic are installed ([PR #166](#)).

3.2.2 NbClassic Developer FAQ

1. Where should I submit my issue?

The Jupyter Notebook 6.4.x will only be taking security fixes, you can follow these guidelines to [report a vulnerability](#).

For Notebook 6.5.x, as it is intended to be end of life and will receive only bug and security fixes, issues of this type in the frontend should be reported in the [jupyter/nbclassic](#) repository, bug and security issues for the server can be submitted in the [jupyter/notebook](#) repository.

Generally, user interface issues dealing with the nbclassic package can be submitted to the [jupyter/nbclassic](#) repository, while server issues can be reported to the [jupyter_server/jupyter_server](#) repository.

Notebook 7.x issues would require closer consideration as they could be reported in the [jupyter/notebook](#), [jupyterlab/jupyterlab](#), or [jupyter_server/jupyter_server](#) repositories depending on the issue.

The Jupyter Community appreciates your efforts in making sure your issue is submitted to the correct project. There are many projects within the Jupyter ecosystem which can mean some issues are best suited for repositories different than those in which they may have been opened. In the case that the issue belongs in a different repository, we can use the [MeeseeksDev bot](#) to move the issue to the appropriate repository so long as the target repository is included in the [allowed organization list](#) in

the MeeseeksDev codebase. You can see a list of the Github links to Jupyter organizations, and the different Jupyter projects under each, in the [Jupyter Community](#) page.

3.2.3 Known issues

Bellow are some known bugs and issues with the NbClassic project. These are items that may be of particular interest to users migrating from notebook to nbclassic.

1. [#140 Error using jupyter_nbextensions_configurator with nbclassic](#) is a known issue with partial fix [Support nbclassic while updating the static path](#) pending to be merged into the [Jupyter-contrib/jupyter_nbextensions_configurator](#) repository.

Once a release with this fix is available, users will be able to activate the extension with the following commands:

```
$ pip install 'jupyter_nbextensions_configurator @ git+https://github.com/
↪datalayer-externals/jupyter-notebook-configurator.git@fix/nbclassic
↪#egg=jupyter_nbextensions_configurator'
$ jupyter nbclassic-extension install --sys-prefix --py jupyter_nbextensions_
↪configurator --overwrite
$ jupyter nbclassic-extension enable --sys-prefix --py jupyter_nbextensions_
↪configurator
$ jupyter nbclassic-serverextension enable --sys-prefix --py jupyter_
↪nbextensions_configurator
```

3.3 Developer FAQ

1. How do I install a prerelease version such as a beta or release candidate?

```
python -m pip install nbclassic --pre --upgrade
```

3.4 My Notebook

```
[1]: def foo():
     ↪ return "foo"
```

```
[2]: def has_ip_syntax():
     ↪ listing = !ls
     ↪ return listing
```

```
[4]: def whatsmyname():
     ↪ return __name__
```

3.5 Other notebook

This notebook just defines bar

```
[2]: def bar(x):  
      return "bar" * x
```

Additional information about contributing to and developing NbClassic.