
euporie

Josiah Outram Halstead

Mar 20, 2022

CONTENTS:

1	Install	3
2	Usage	5
3	Documentation	7
4	Features	9
5	Compatibility	11
5.1	Gallery	11
5.2	Installation	14
5.3	Usage	17
5.4	Configuration	19
5.5	Key Bindings	26
5.6	Changelog	32
5.7	Related Projects	36
5.8	euporie	36
6	Indices and tables	169
	Python Module Index	171
	Index	173

Euporie is a terminal app for running and editing Jupyter notebooks.

The text-based interface is inspired by JupyterLab / Jupyter Notebook, and runs entirely in the terminal.

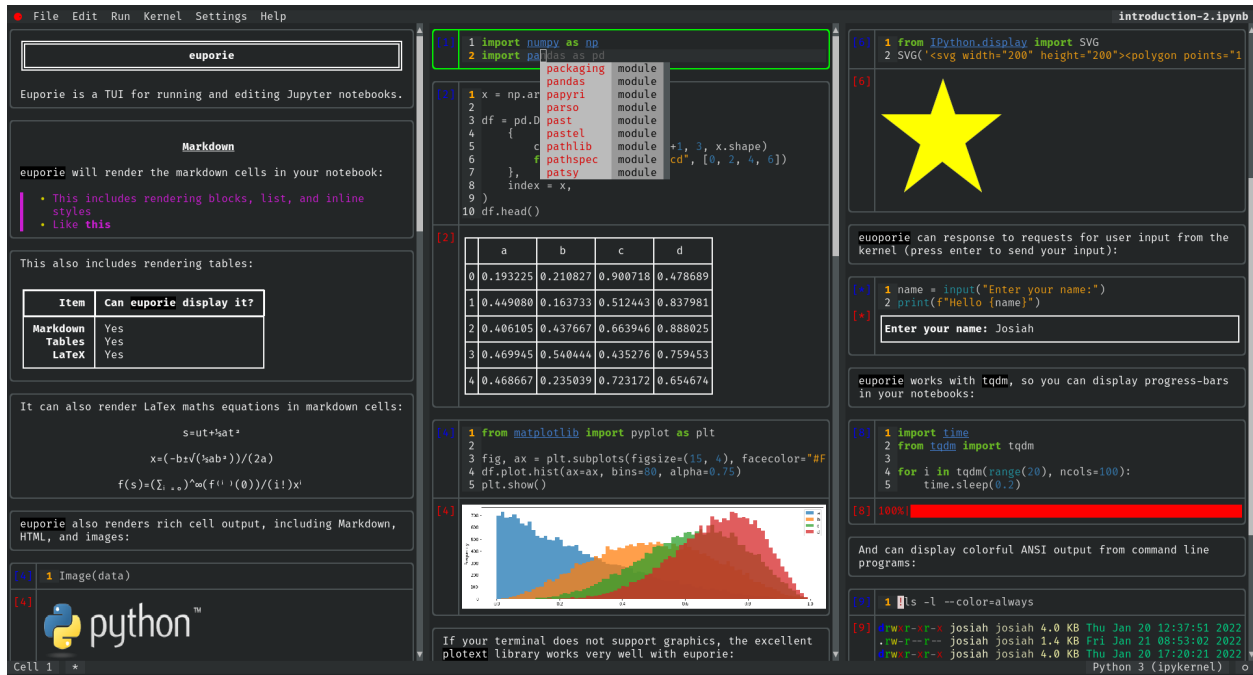


Fig. 1: [View more screenshots here](#)

INSTALL

You can install euporie with [pipx](#) (recommended) or `pip`:

```
$ pipx install euporie  
$ # OR  
$ python -m pip install --user euporie
```

You can also try euporie online [here](#).

USAGE

Open a notebook by passing the notebook's file path as a command line argument:

```
$ euporie notebook.ipynb
```

Alternatively, launch `euporie` and open a notebook file by selecting “Open” from the file menu (*Ctrl+o*).

For more information about the available command line flags, run:

```
$ euporie --help
```


DOCUMENTATION

View the online documentation at: <https://euporie.readthedocs.io/>

The code is available on GitHub at: <https://github.com/jouuha/euporie>

FEATURES

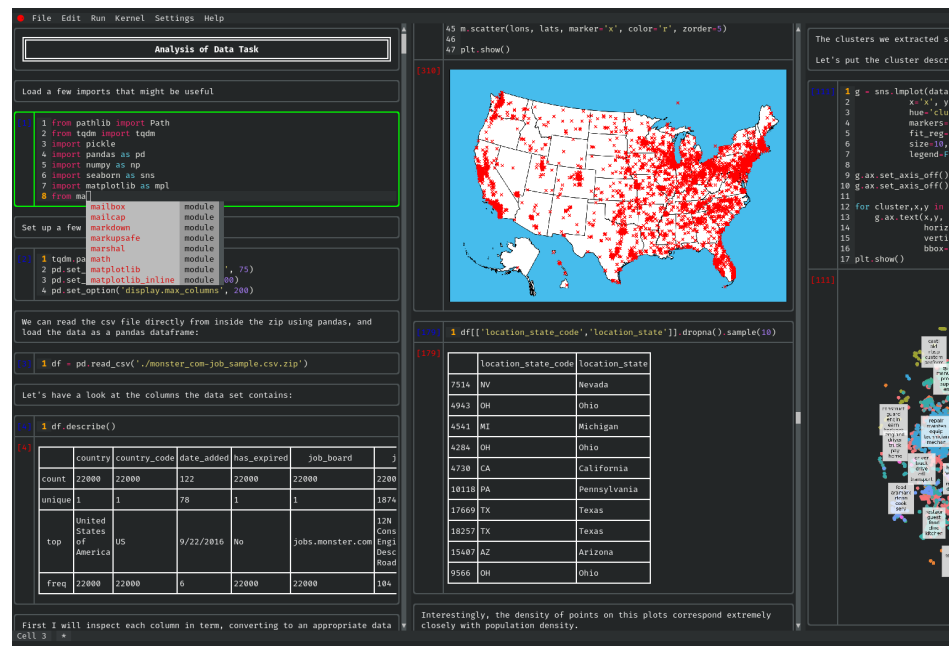
- Edit and run notebooks in the terminal
- Displays rich cell outputs, including markdown, tables, images, LaTeX, HTML, SVG, & PDF
- Print formatted notebooks to the terminal or pager
- Open multiple notebooks side-by-side
- Highly configurable
- Code completion
- Line completions from history
- Display contextual help
- Automatic code formatting

COMPATIBILITY

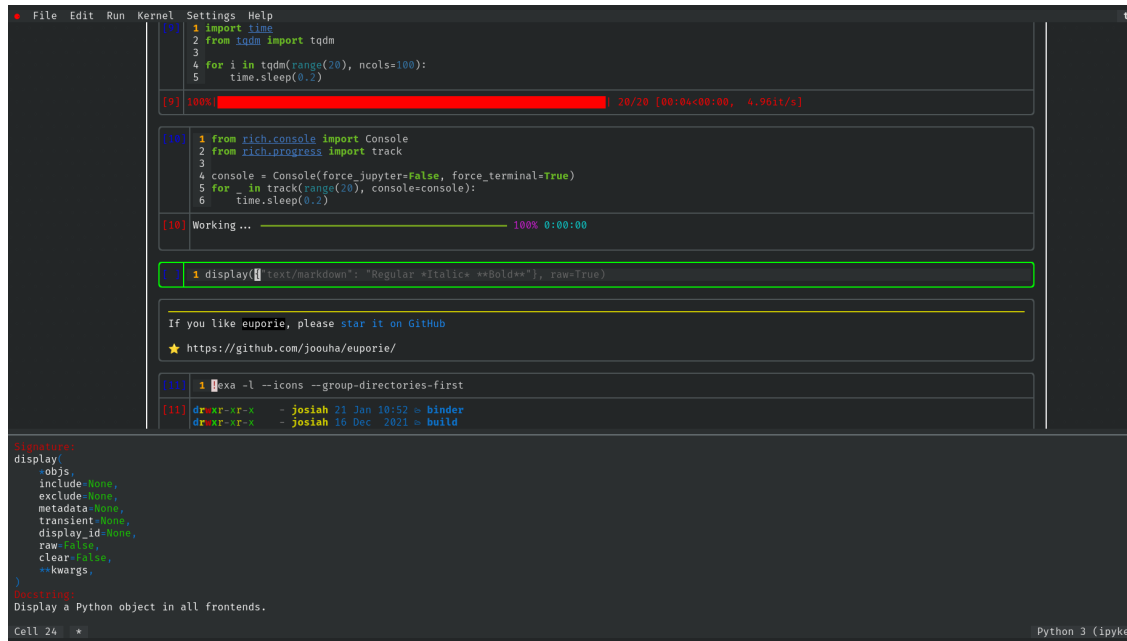
Euporie requires Python 3.8 or later. It works on Linux, Windows and MacOS

5.1 Gallery

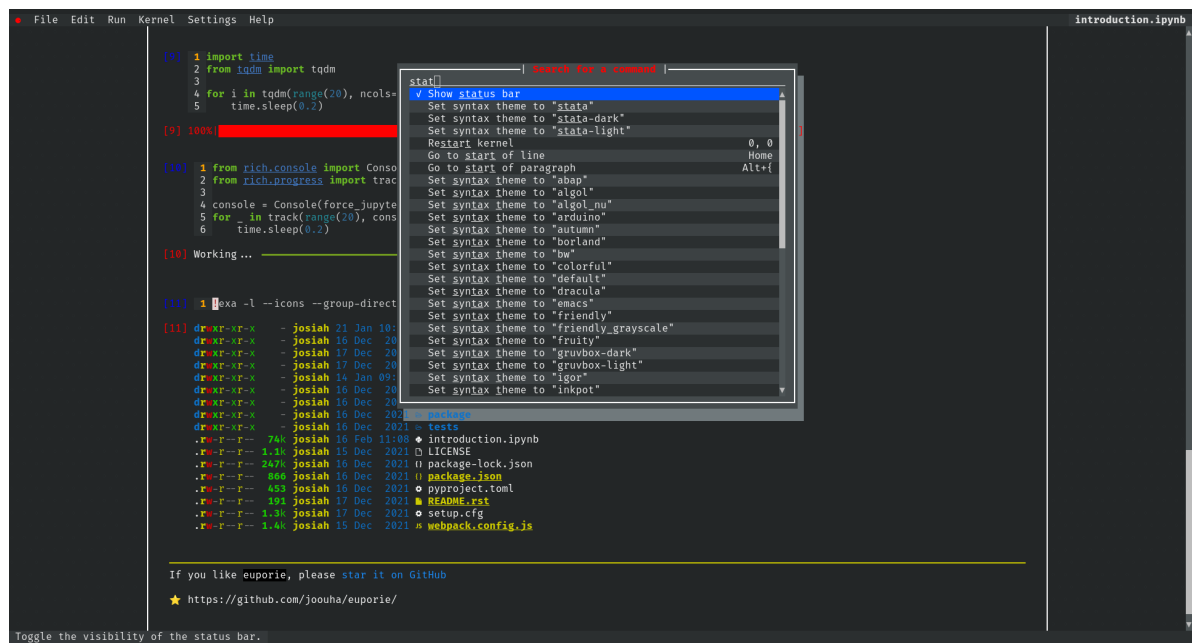
This page showcases screenshots demonstrating some of the features of euporie



Displaying multiple notebooks side-by-side:



Displaying contextual help:



Command Palette:

File Edit Run Kernel Settings Help

Euporie can render markdown tables:

Item	Can euporie display it?
Markdown	Yes
Tables	Yes
LaTeX	Yes

It can also render LaTeX maths equations in markdown cells:

$$s = t \cdot u + \frac{a \cdot t^2}{2}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$f(s) = \sum_{i=0}^{\infty} \frac{1}{x} \cdot \frac{f \cdot \theta}{i!}$$

$$1 = 0$$

```
[2] from IPython.display import *
    display({"text/markdown": "Regular *Italic* **Bold**"}, raw=True)

[2] Regular Italic Bold
```

Cell 3

Display LaTeX formulae in markdown cells with Sympy:

File Edit Run Kernel Settings Help

```
[2] 1 from sympy import *
    2 import matplotlib.pyplot as plt
    3
    4 init_printing(use_unicode=True, use_latex=False)

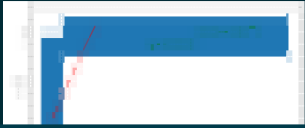
[3] 1 width = 20
    2 length = 210
    3
    4 x = symbols('x')
    5 equation = Eq((length - x - width+1.5)**2 + width**2, (2 + width)**2 + (x + width/2)**2)
    6 equation

[4] 32400.0*(1 - 0.00555555555555556*x)**2 + 400 = 100.0*(0.1*x + 1)**2 + 1600

[4] 1 x, = solve(equation, x)
    2 x

[4] 81.8421052631579

[11] 1 fig, ax = plt.subplots(figsize=(8,8), facecolor="#FFF")
    2
    3 points = [(length-width+1.5, width/2), (x, -width/2)]
    4 plt.plot(*zip(*points), marker='x', color='g')
    5
    6 points = [(width+1.5, width/2), (-width/2, -x)]
    7 plt.plot(*zip(*points), marker='x', color='r')
    8
    9 ax.add_patch(plt.Rectangle((0,0), length, width))
   10 ax.add_patch(plt.Rectangle((0,0), length, width))
   11 ax.add_patch(plt.Rectangle((0,0), -width, -length))
   12
   13 ax.autoscale()
   14 ax.set_aspect('equal')
```

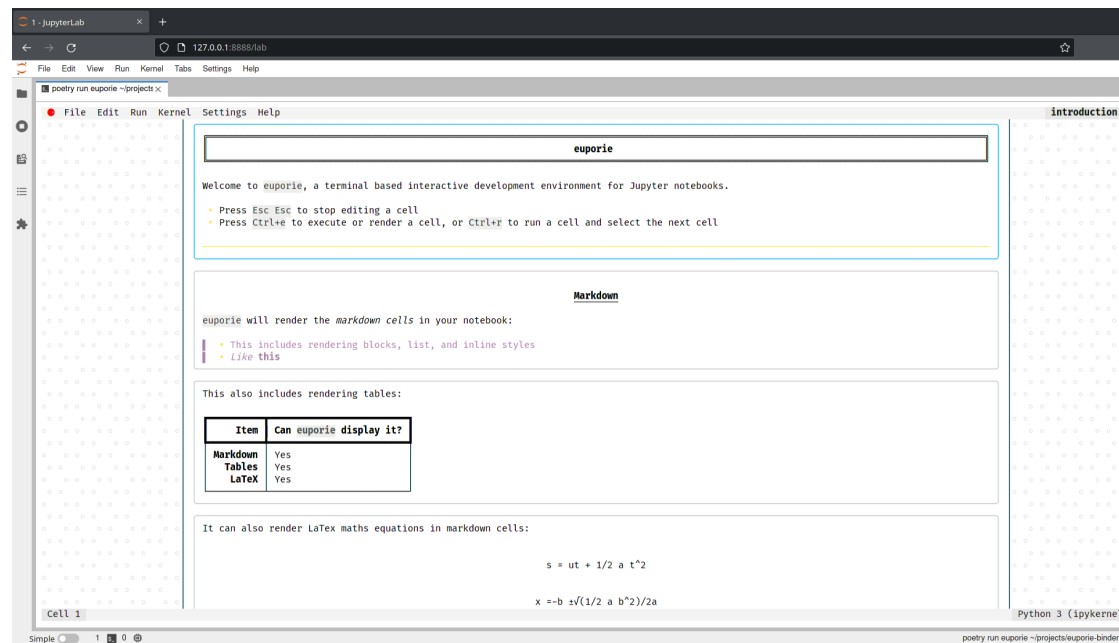


Cell 3

Adapting to terminal colour theme:



Running on Windows:



Running inside JupyterLab:

5.2 Installation

Euporie is on [pypi](#), so can be installed using [pip](#) or [pipx](#).

To install euporie globally, run:

```
$ pipx install euporie
```

To install inside a virtualenv, run:

```
$ pip install euporie
```

If you want to try the latest and potentially unstable unreleased changes, you can install euporie from git:

```
$ pipx install git+https://github.com/joouha/euporie.git
```

5.2.1 Optional Dependencies

Euporie supports a wide range of rendering methods in order to get your notebooks looking as nice as possible in the terminal. The following section lists the various rendering methods available, and details what needs to be installed for them to be used.

Images

Euporie will attempt to render images in the best possible way it can.

Note: `timg` is installed as a dependency of euporie and is used to render images as sixels or ansi art. However, euporie will preferentially use an external application if it is installed and is more performant or gives higher quality output.

The following methods will be used if they are available:

Kitty's Terminal Graphics Protocol

If your terminal supports [kitty's terminal graphics protocol](#), euporie will use it to render images.

Currently only the [kitty](#) and [WezTerm](#) terminals support this.

Sixels

If supported by your terminal, euporie can show graphical images in cell outputs using the Sixel graphics protocol. This requires one of the following dependencies:

- **Python packages**

- `timg`
 - `teimpy`

- **External applications**

- `img2sixel`
 - `imagemagick`

Ansi Art

If all else fails, euporie will fall back to using ansi art to display images.

- **Python packages**

- `timg`

- **External applications**

- `chafa`

- `timg`

- `cating`

- `icat`

- `tiv`

- `viu`

- `img2unicode`

- `jp2a`

- `img2txt`

SVG

Euporie can display SVG output by first rasterizing it, for which one of the following is required:

- **Python packages**

- `cairosvg`

- **External applications**

- `imagemagick`

HTML

Euporie will attempt to render HTML outputs. This requires one of the following:

- **Python packages**

- `mtable`

- **External applications**

- `w3m`

- `elinks`

- `lynx`

- `links`

Note: `mtable` will only render HTML tables in cell outputs, so is useful if you are working with dataframes

LaTeX

Euporie will render LaTeX in markdown and cell output using unicode text, using any of the following if they are installed:

- **Python packages**

- flatlatex
- sympy
- pylatexenc
- ipython

- **External applications**

- **dvipng**

Note: flatlatex is a dependency of euporie, so will be installed by default

Note: ipython and **dvipng** are both required to display rendered LaTeX as graphics

5.3 Usage

5.3.1 Command Line Interface

To open a notebook from the command line, launch euporie with the notebook file's path as an argument:

```
$ euporie ./my-notebook.ipynb
```

The following command line arguments are available:

Positional Arguments

<Path ...>

A sequence of notebook file-paths to open

Options

-h, --help

show this help message and exit

--version, -V

Show the version number and exit

--log-file <str>

File path for logs

--debug, --no-debug

Include debug output in logs

--dump, --no-dump

Output formatted file to display or file

--dump-file <Path>

Output path when dumping file

--page, --no-page

Pass output to pager

--run, --no-run
Run the notebook when loaded

--tmux-graphics, --no-tmux-graphics
Enable terminal graphics in tmux (experimental)

--terminal-polling-interval <int>
Time between terminal colour queries

--edit-mode {micro,emacs,vi}
Key-binding mode for text editing

--tab-size <int>
Spaces per indentation level

--run-after-external-edit
Run cells after editing externally

--format-black, --no-format-black
Use black when re-formatting code cells

--format-isort, --no-format-isort
Use isort when re-formatting code cells

--format-ssort, --no-format-ssort
Use ssort when re-formatting code cells

--autofformat, --no-autofformat
Automatically re-format code cells when run

--autocomplete, --no-autocomplete
Provide completions suggestions automatically

--autosuggest, --no-autosuggest
Provide line completion suggestions

--autoinspect, --no-autoinspect
Display contextual help automatically

--expand, --no-expand
Use the full width to display notebooks

--max-notebook-width <int>
Maximum width of notebooks

--show-status-bar, --no-show-status-bar
Show the status bar

--color-scheme {default,inverse,light,dark}
The color scheme to use

--background-pattern {0,1,2,3,4,5}, --bg-pattern {0,1,2,3,4,5}
The background pattern to use

--background-character <str>, --bg-char <str>
Character for background pattern

--background-color <str>, --bg-color <str>
Color for background pattern

--show-cell-borders, --no-show-cell-borders
Show or hide cell borders.

--line-numbers, --no-line-numbers

Show or hide line numbers

--syntax-theme <str>

Syntax highlighting theme

5.4 Configuration

5.4.1 Configuring Euporie

Euporie has a range of configurable options which affect euporie's behaviour and appearance.

Options are validated at application startup, so if an option is set to an invalid value, it will be ignored.

The options can be set in three different ways:

Command Line Arguments

Euporie can be configured by passing flags (and sometimes values) on the command line when euporie is launched. The flags for each configuration option and the allowed values are listed in [Configuration Options](#).

```
$ euporie --color-scheme=light --no-show-cell-borders --expand notebook.ipynb
```

Options set on the command line will override those set via an environment variable, in the configuration file, and the default values.

Environment Variables

Euporie can be configured by setting environment variables. Each option can be set by assigning a value to an environment variable with uppercase option name, prefixed with EUPORIE_.

```
$ EUPORIE_COLOR_SCHEME=light EUPORIE_SHOW_CELL_BORDERS= EUPORIE_EXPAND=True euporie_
↩ notebook.ipynb
```

Setting boolean values to an empty string will cause them to evaluate to `False`.

Options set in via an environment variable will override those set in the configuration file and the default values.

Configuration File

Euporie can be configured using a JSON configuration file. The file takes the form of *key: value* pairs, where the key is one of the options listed in [Configuration Options](#)

Warning: The configuration file is read when euporie is launched, and modifying options from the *Settings* menu in euporie will cause the configuration file to be updated. Thus, any changes made to the configuration file while euporie is running may be lost, so this is not recommended.

Example

```
{
  "color_scheme": "light",
  "syntax_theme": "native",
  "show_cell_borders": false,
  "expand": true
}
```

File Location The location of Euporie's configuration file depends on your operating system:

Linux	\$XDG_CONFIG_HOME/euporie/config.json
Mac OS	~/Library/Application Support/<euporie>/config.json
Windows	%APPDATA%\Local\euporie\config.json

If the file cannot be parsed as valid JSON, the file will be ignored.

Options set in the configuration file will override the default values.

5.4.2 Configuration Options

All available configuration options are listed below:

version

flags `--version` or `-V`

description Show the version number and exit

If set, euporie will print the current version number of the application and exit. All other configuration options will be ignored.

Note: This cannot be set in the configuration file or via an environment variable

log_file

flags `--log-file`

environment variable EUPORIE_LOG_FILE

default ''

type string

description File path for logs

When set to a file path, the log output will be written to the given path. If no value is given output will be sent to the standard output.

debug

flags `--debug`

environment variable EUPORIE_DEBUG

default False

type boolean

description Include debug output in logs

When set, logging events at the debug level are emitted.

dump

flags `--dump`

environment variable EUPORIE_DUMP

default False

type `boolean`

description Output formatted file to display or file

When set, the formatted output will be written to the the output file path given by *dump_file* (standard output by default).

dump_file

flags `--dump-file`

environment variable EUPORIE_DUMP_FILE

type `string`

description Output path when dumping file

When set to a file path, the formatted output will be written to the given path. If no value is given (or the default “-” is passed) output will be printed to standard output.

page

flags `--page`

environment variable EUPORIE_PAGE

default False

type `boolean`

description Pass output to pager

Whether to pipe output to the system pager when using `--dump`.

run

flags `--run`

environment variable EUPORIE_RUN

default False

type `boolean`

description Run the notebook when loaded

If set, notebooks will be run automatically when opened, or if dumping output, notebooks will be run before being output.

tmux_graphics

flags `--tmux-graphics`

environment variable EUPORIE_TMUX_GRAPHICS

default False

type `boolean`

description Enable terminal graphics in `tmux` (experimental)

If set, terminal graphics will be used if `tmux` is running by performing terminal escape sequence pass-through. You must restart euporie for this to take effect.

Warning: Terminal graphics in `tmux` is experimental, and is not guaranteed to work. Use at your own risk!

terminal_polling_interval

flags `--terminal-polling-interval`

environment variable `EUPORIE_TERMINAL_POLLING_INTERVAL`

default `0`

type `integer`

description Time between terminal colour queries

Determine how frequently the terminal should be polled for changes to the background / foreground colours. Set to zero to disable terminal polling.

edit_mode

flags `--edit-mode`

environment variable `EUPORIE_EDIT_MODE`

default `'micro'`

type `string`

options `['micro', 'emacs', 'vi']`

description Key-binding mode for text editing

Key binding mode to use when editing cells.

tab_size

flags `--tab-size`

environment variable `EUPORIE_TAB_SIZE`

default `4`

type `integer`

description Spaces per indentation level

The number of spaces to use per indentation level. Should be set to 4.

run_after_external_edit

flags `--run-after-external-edit`

environment variable `EUPORIE_RUN_AFTER_EXTERNAL_EDIT`

default `False`

type `boolean`

description Run cells after editing externally

Whether to execute a cell immediately after editing in `$EDITOR`.

format_black

flags *--format-black*

environment variable EUPORIE_FORMAT_BLACK

default True

type *boolean*

description Use black when re-formatting code cells

Whether to use black when reformatting code cells.

format_isort

flags *--format-isort*

environment variable EUPORIE_FORMAT_ISORT

default True

type *boolean*

description Use isort when re-formatting code cells

Whether to use isort when reformatting code cells.

format_ssort

flags *--format-ssort*

environment variable EUPORIE_FORMAT_SSORT

default True

type *boolean*

description Use ssort when re-formatting code cells

Whether to use ssort when reformatting code cells.

autoformat

flags *--autoformat*

environment variable EUPORIE_AUTOFORMAT

default False

type *boolean*

description Automatically re-format code cells when run

Whether to automatically reformat code cells before they are run.

autocomplete

flags *--autocomplete*

environment variable EUPORIE_AUTOCOMPLETE

default False

type *boolean*

description Provide completions suggestions automatically

Whether to automatically suggestion completions while typing in code cells.

autosuggest

flags *--autosuggest*

environment variable EUPORIE_AUTOSUGGEST

default True

type `boolean`

description Provide line completion suggestions

Whether to automatically suggestion line content while typing in code cells.

autoinspect

flags `--autoinspect`

environment variable EUPORIE_AUTOINSPECT

default False

type `boolean`

description Display contextual help automatically

Whether to automatically display contextual help when navigating through code cells.

expand

flags `--expand`

environment variable EUPORIE_EXPAND

default False

type `boolean`

description Use the full width to display notebooks

Whether the notebook page should expand to fill the available width

max_notebook_width

flags `--max-notebook-width`

environment variable EUPORIE_MAX_NOTEBOOK_WIDTH

default 120

type `integer`

description Maximum width of notebooks

The maximum width at which to display a notebook.

show_status_bar

flags `--show-status-bar`

environment variable EUPORIE_SHOW_STATUS_BAR

default True

type `boolean`

description Show the status bar

Whether the status bar should be shown at the bottom of the screen.

color_scheme

flags `--color-scheme`

environment variable EUPORIE_COLOR_SCHEME

default 'default'

type string

options ['default', 'inverse', 'light', 'dark']

description The color scheme to use

The color scheme to use: *auto* means euporie will try to use your terminal's color scheme, *light* means black text on a white background, and *dark* means white text on a black background.

background_pattern

flags *--background-pattern* or *--bg-pattern*

environment variable EUPORIE_BACKGROUND_PATTERN

default 2

type integer

options [0, 1, 2, 3, 4, 5]

description The background pattern to use

The background pattern to use when the notebook is narrower than the available width. Zero mean no pattern is used.

background_character

flags *--background-character* or *--bg-char*

environment variable EUPORIE_BACKGROUND_CHARACTER

default '.'

type string

description Character for background pattern

The character to use when drawing the background pattern.

Recommened characters include: ".", ",", ":", ";", "\\", "<", ">", "<>", "<=", ">=", "<=>".

background_color

flags *--background-color* or *--bg-color*

environment variable EUPORIE_BACKGROUND_COLOR

default ''

type string

description Color for background pattern

The color to use for the background pattern.

show_cell_borders

flags *--show-cell-borders*

environment variable EUPORIE_SHOW_CELL_BORDERS

default False

type boolean

description Show or hide cell borders.

Whether cell borders should be drawn for unselected cells.

line_numbers

flags *--line-numbers*

environment variable EUPORIE_LINE_NUMBERS

default True

type *boolean*

description Show or hide line numbers

Whether line numbers are shown by default.

syntax_theme

flags *--syntax-theme*

environment variable EUPORIE_SYNTAX_THEME

default 'default'

type string

description Syntax highlighting theme

The name of the pygments style to use for syntax highlighting.

files

environment variable EUPORIE_FILES

default []

type array

description List of file names to open

A list of file paths to open when euporie is launched.

5.5 Key Bindings

5.5.1 Editing Modes

The key-bindings used when editing a cell are determined by the *edit_mode* configuration variable. This can be set to *micro*, *emacs* or *vim* to use key-bindings in the style of the respective text editor.

Note: The *vim* and *emacs* key-bindings are defined by the *prompt_toolkit* package and are largely untested, so your results may vary!

5.5.2 Running Cells

Cells can be run using **Ctrl+Enter**, or **Shift+Enter** to run and select the next cell, as is the case in [JupyterLab](#).

However, most terminals do not distinguish between **Enter**, **Ctrl+Enter** & **Shift+Enter** by default, meaning that you have to use alternative key-bindings in euporie to run cells.

Fortunately it is possible to configure many terminals such that these key-bindings can be used, as outlined below.

Note: There are two commonly used formats of escape sequences which can be used to distinguish these key-bindings: **FK-27** and **CSI-u**. The instructions below implement the CSI-u style, but euporie will recognise either.

WezTerm

Update your `$HOME/.config/wezterm/wezterm.lua` file to include the following:

```
local wezterm = require 'wezterm';

return {
  -- ...

  keys = {
    {key="Enter", mods="CTRL", action=wezterm.action{SendString="\x1b[13;5u"}},
    {key="Enter", mods="SHIFT", action=wezterm.action{SendString="\x1b[13;2u"}},
  },
}
```

Kitty

Add the following to your `:file:$HOME/.config/kitty/kitty.conf` file:

```
map ctrl+enter send_text normal,application \x1b[13;5u
map shift+enter send_text normal,application \x1b[13;2u
```

Foot

Foot supports XTerm's **K27** format, so does not require any additional configuration.

XTerm

You can add the following lines to your `$HOME/.Xresources` file, which enables **CSI-u** escape sequences.

```
*vt100.modifyOtherKeys: 1
*vt100.formatOtherKeys: 1
```

Windows Terminal

You can add the key-bindings to your `settings.json` file:

```
{
  // ...

  "keybindings": [
    { "command": { "action": "sendInput", "input": "\u001b[13;5u" }, "keys": "ctrl+enter" },
    { "command": { "action": "sendInput", "input": "\u001b[13;2u" }, "keys": "shift+enter" }
  ]
}
```

Alacritty

You can define the key-binding in your `$HOME/.config/alacritty/alacritty.yml` file as follows:

```
key_bindings:
- { key: Return, mods: Control, chars: "\x1b[13;5u" }
- { key: Return, mods: Shift,   chars: "\x1b[13;2u" }
```

Konsole

In the menu, navigate to *Settings* → *Edit Current Profile*, then select *Keyboard* → *Edit*.

Change the existing entry for *Return+Shift* to *Return+Shift+Ctrl* (or whatever you prefer), then add the following entries:

Key combination	Output
Return+Ctrl	E[13;5u
Return+Shift	E[13;2u

5.5.3 Default Key bindings Reference

The following lists outline the default key-bindings used in euporie:

App

Keys	Command
Ctrl+n	Create a new file.
Ctrl+o	Open a file.
Ctrl+w	Close the current file.
Ctrl+q	Quit euporie.
Ctrl+Pagedown	Switch to the next tab.
Ctrl+Pageup	Switch to the previous tab.
Tab	Focus the next control.
Ctrl+i	
Shift+Tab	Focus the previous control.
Ctrl+@	Shows the command palette.

Config

Keys	Command
l	Toggle the visibility of line numbers.
w	Toggle whether cells should extend across the full width of the screen.

Notebook

Keys	Command
Ctrl+s	Save the current notebook.
Ctrl+Enter Ctrl+e	Run or render the current cells.
Shift+Enter Ctrl+r	Run or render the current cells and select the next cell.
Ctrl+Alt+Enter	Run or render the current cells and insert a new cell below.
a	Add a new cell above the current.
b	Add a new cell below the current.
d, d	Delete the current cells.
x	Cut the current cells.
c	Copy the current cells.
v	Paste the previously copied cells.
M	Merge the selected cells.
I, I	Interrupt the notebook's kernel.
0, 0	Restart the notebook's kernel.
[Scroll the page up a line.
]	Scroll the page down a line.
{	Scroll the page up 5 lines.
}	Scroll the page down 5 lines.
Home Ctrl+Up	Select the first cell in the notebook.
Pageup	Go up 5 cells.
Up k	Go up one cell.

continues on next page

Table 1 – continued from previous page

Keys	Command
Down j	Select the next cell.
Pagedown	Go down 5 cells.
End Ctrl+Down	Select the last cell in the notebook.
Ctrl+a	Select all cells in the notebook.
Shift+Up K	Go up one cell.
Shift+Down J	Go up one cell.
m	Change selected cells to markdown cells.
y	Change selected cells to code cells.
r	Change selected cells to raw cells.
F	Automatically reformat all code cells in the notebook.

Cell

Keys	Command
e	Edit cell in \$EDITOR.
Enter Ctrl+m	Enter cell edit mode.
Escape Alt+Escape	Exit cell edit mode.
Ctrl+\	Split the current cell at the cursor position.
f	Format the selected code cells.

Pager

Keys	Command
Escape q	Close the pager.

Suggestion

Keys	Command
Right Ctrl+f	Accept suggestion.
Alt+f	Fill partial suggestion.

Micro edit mode

Keys	Command
Insert	Toggle overwrite when using micro editing mode.
Backspace Ctrl+h Backspace Ctrl+h	Delete the character behind the cursor.
Ctrl+Left Alt+b	Move back to the start of the current or previous word.
Ctrl+Right Alt+f	Move forward to the end of the next word.
Ctrl+Up Ctrl+Home	Move to the start of the buffer.
Ctrl+Down Ctrl+End	Move to the end of the buffer.
Pagedown	Scroll page down.
Pageup	Scroll page up.
Left	Move back a character, or up a line.
Right	Move forward a character, or down a line.
Home Alt+Left Alt+a	Move the cursor to the start of the line.
End Alt+Right Alt+e	Move the cursor to the end of the line.
Alt+{	Move the cursor to the start of the current paragraph.
Alt+}	Move the cursor to the end of the current paragraph.
Ctrl+/ Ctrl+_	Comments or uncomments the current or selected lines.
"	Wraps the current selection with: ""
'	Wraps the current selection with: ''
()	Wraps the current selection with: ()
{ }	Wraps the current selection with: {}
[]	Wraps the current selection with: []
\`	Wraps the current selection with: ``
*	Wraps the current selection with: **
_	Wraps the current selection with: __
Ctrl+d	Duplicate the current line.
Ctrl+d	Duplicate the current line.
Ctrl+v	Paste the clipboard contents, replacing any current selection.
Ctrl+c	Adds the current selection to the clipboard.
Ctrl+x	Removes the current selection and adds it to the clipboard.
Ctrl+k	Removes the current line adds it to the clipboard.
Alt+Up	Move the current or selected lines up by one line.
Alt+Down	Move the current or selected lines down by one line.

continues on next page

Table 2 – continued from previous page

Keys	Command
Enter Ctrl+m	Accept an input.
Enter Ctrl+m	Insert a new line, replacing any selection and indenting if appropriate.
Tab Ctrl+i	Inndent the current or selected lines.
Shift+Tab	Unindent the current or selected lines.
Backspace Ctrl+h	Unindent the current or selected lines.
F4	Toggle the case of the current word or selection.
Ctrl+z	Undo the last edit.
Ctrl+y	Redo the last edit.
Ctrl+a	Select all text.
Shift+Tab	Displays contextual help.

5.6 Changelog

Notable changes to this project will be documented in this file.

5.6.1 Upcoming

5.6.2 1.3.1 - 2022-03-20

Fixed

- Fix notebook dumping regression
-

5.6.3 1.3.0 - 2022-03-19

Added

- Added ability to select multiple notebook cells
- Added ability to merge multiple cells
- Added ability to split cells
- Added commands to move cells up and down

Changed

- Expanded run, cut, copy, paste commands to work with multiple cells
- Changing cell type affects all selected cells
- Formatting cells formats all selected code cells

Fixed

- Fix recursion bug when editing a cell when multiple cells are selected
-

5.6.4 1.2.2 - 2022-03-17**Fixed**

- Fix zero-division error if scrolling a window with less content than its height
-

5.6.5 1.2.1 - 2022-03-17**Added**

- Make terminal colour polling timeout configurable
- Add ability to focus and scroll the inspection pane
- Add inspection pane key-bindings

Changed

- Use improved scrollbars with mouse support (if [PR #1587](#) is merged)

Fixed

- Ensure opening command palette does not show an error if it is opened before it has loaded
 - Fixed bug where nothing would be focused after a completion if pager was opened
-

5.6.6 1.2.0 - 2022-03-06**Added**

- Inspection pane for showing contextual help (summon with `Shift+Tab` in a code cell)
- Add ability to use `isort` and `ssort` when formatting code cells
- Make terminal colour change detection interval configurable

Fixed

- Indent if cursor in leading whitespace rather than suggest complete
 - No longer continue indenting on subsequent newlines after a colon
-

5.6.7 1.1.0 - 2022-03-04

Added

- Obey the NO_COLOR environment variable (<https://no-color.org/>)
- Graphic background now follows theme color not terminal color
- Add command palette (summoned with `Ctrl+space`)
- Add experimental support for terminal graphics from within `tmux`
- Add support for displaying images using the `iTerm inline images` protocol
- New terminal graphic rendering system to work with new scrolling method
- Allow displaying pager outputs (e.g. when using `print??` `ipython` syntax)
- Display PDF outputs
- Add option to automatically format code cells with black
- Experimental support for terminal graphics in `tmux`

Changed

- New notebooks scrolling method which improves scrolling performance
- New data conversion system to replace the output rendering system

Fixed

- Prevent output from the kernel subprocess being printed and breaking the display
 - Display tabs in ansi output correctly
 - Prevent hidden cell borders show up when syntax theme is changed
 - Graphics now use theme background colour rather than terminal background colour
 - Fixed occasional error when uncommenting a block of text
 - Prevent terminal graphics obscuring dialogs
 - Run all cells now works as expected in TUI mode
 - Restore clipboard functionality
-

5.6.8 1.0.0 - 2020-02-03

Added

- Added documentation
- Add shortcut key (c-/) to toggle line commenting
- Improved terminal feature detection
- Add ability to render LaTeX using `sympy`
- Add new terminal graphics system, which makes displaying using sixels / kitty graphics more reliable
- Add support for LaTeX equations in markdown using flatlatex
- Markdown tables no longer expand to the full width of the display
- Show menu item descriptions in the statusbar
- Add keyboard shortcuts to the menu
- Allow status bar to be hidden
- Allow wrapping selection in quotes or brackets
- Display keyboard shortcuts on menus, and better looking menus
- Add *micro* editor style key-bindings as the default
- Add centralized command system
- Configurable cell border visibility
- Read notebook language configuration from the kernel
- Configurable colorschemes
- Support user input with `input()`
- Support asynchronous cell output (à la `akernel`)
- Vastly more responsive completion & history requests
- Major code restructure
- Much improved scrollbar
- Added ability to view the logs in a tab
- Introduced the concept of “tabs” (tabs are only displayed vertically tiled for now)
- Automatic suggestions from kernel history
- Completion type annotations in the completion menu
- Added ability to automatically run notebooks with `-run` command line flag
- Allow changing background colour
- Add `chafa` as an image renderer

Fixed

- Fix issue where clicking on cells failed to focus them

5.7 Related Projects

5.7.1 Notebook Editors

nbterm An alternative effort sponsored by QuantStack

nbtermix A fork of nbterm

jpterm Jupyter in the terminal

5.7.2 Notebook Viewers

nbpreview An excellent terminal viewer for Jupyter notebooks, with image and LaTeX support

ipynbviewer Outputs Jupyter Notebook in the terminal in a human readable format (supports images)

nbtui A cli Jupyter notebook viewer with support for kitty's terminal graphics protocol

jupyterui A cli Jupyter notebook viewer

jut Another cli Jupyter notebook viewer

juoview View jupyter notebooks from terminal

jcat A command line tool for viewing notebook files in terminal (written in C++)

ipynbat A terminal jupyter notebook viewer written in rust

nbcats View contents of Jupyter notebooks in the terminal with syntax highlighting

nbv Quickly preview jupyter notebooks in terminal

ipynb-term A simple script to read jupyter notebooks in the terminal

Read-Jupyter-Notebook Check the version or content of a Jupyter Notebook file from the terminal

5.8 euporie

This package defines the euporie application and its components.

Modules

<code>euporie.app</code>	Defines euporie's application classes.
<code>euporie.box</code>	Define box border constants.
<code>euporie.cell</code>	Defines a cell object with input are and rich outputs, and related objects.
<code>euporie.commands</code>	Defines a centralized command system.
<code>euporie.completion</code>	Contains the main class for a notebook file.
<code>euporie.config</code>	Defines a configuration class for euporie.

continues on next page

Table 3 – continued from previous page

<code>euporie.containers</code>	Contains the <i>ScrollingContainer</i> class, which renders children on the fly.
<code>euporie.convert</code>	Sub-module concerned with the conversion of data formats.
<code>euporie.filters</code>	Defines common filters.
<code>euporie.format</code>	Contains functions to automatically format code cell input.
<code>euporie.kernel</code>	Contains the main class for a notebook file.
<code>euporie.key_binding</code>	Defines key-bindings for the application.
<code>euporie.keys</code>	Defines KeyBindings wrapper which keeps track of key binding descriptions.
<code>euporie.log</code>	Initiate logging for euporie.
<code>euporie.margins</code>	Contains margins.
<code>euporie.markdown</code>	Defines markdown extensions for commonmark and rich.
<code>euporie.menu</code>	Defines the application's menu bar and them menu's contents.
<code>euporie.notebook</code>	Contains the main class for a notebook file.
<code>euporie.output</code>	Defines an output container and the controls used within in.
<code>euporie.palette</code>	Contains the command palette container.
<code>euporie.scroll</code>	Contains the <i>ScrollingContainer</i> class, which renders children on the fly.
<code>euporie.style</code>	Style related functions.
<code>euporie.suggest</code>	Suggests line completions from kernel history.
<code>euporie.tab</code>	Contains the main class for a notebook file.
<code>euporie.terminal</code>	Contains classes related to querying terminal features.
<code>euporie.text</code>	Contains updated ANSI parsing and Formatted Text processing.

5.8.1 euporie.app

Defines euporie's application classes.

Modules

<code>euporie.app.base</code>	Contains the main Application class which runs euporie.
<code>euporie.app.current</code>	Defines functions which allow access to typed version of the current application.
<code>euporie.app.dump</code>	Concerns dumping output.
<code>euporie.app.tui</code>	A text base user interface for euporie.

euporie.app.base

Contains the main Application class which runs euporie.

Classes

<code>EuporieApp(**kwargs)</code>	The base euporie application class.
-----------------------------------	-------------------------------------

euporie.app.base.EuporieApp

class `euporie.app.base.EuporieApp(**kwargs: Any)`

The base euporie application class.

This subclasses the `prompt_toolkit.application.Application` class, so application wide methods can be easily added.

class `euporie.app.base.EuporieApp(**kwargs: Any)`

Bases: `prompt_toolkit.application.application.Application`

The base euporie application class.

This subclasses the `prompt_toolkit.application.Application` class, so application wide methods can be easily added.

add_float(`float_container: Float`) → `None`

Adds a float to the application.

async cancel_and_wait_for_background_tasks() → `None`

Cancel all background tasks, and wait for the cancellation to be done. If any of the background tasks raised an exception, this will also propagate the exception.

(If we had nurseries like Trio, this would be the `__aexit__` of a nursery.)

property cell: `Optional[InteractiveCell]`

Return the currently active cell.

cleanup_closed_tab(`tab: euporie.tab.Tab`) → `None`

Remove a tab container from the current instance of the app.

Parameters `tab` – The closed instance of the tab container

close_tab(`tab: Optional[Tab] = None`) → `None`

Closes a notebook tab.

Parameters `tab` – The instance of the tab to close. If `None`, the currently selected tab will be closed.

property color_depth: `prompt_toolkit.output.color_depth.ColorDepth`

The active ColorDepth.

The current value is determined as follows:

- If a color depth was given explicitly to this application, use that value.
- Otherwise, fall back to the color depth that is reported by the Output implementation. If the Output class was created using `output.defaults.create_output`, then this value is coming from the `$PROMPT_TOOLKIT_COLOR_DEPTH` environment variable.

cpr_not_supported_callback() → `None`

Called when we don't receive the cursor position response in time.

create_background_task(*coroutine: Awaitable[None]*) → `_asyncio.Task`

Start a background task (coroutine) for the running application. When the *Application* terminates, unfinished background tasks will be cancelled.

If `asyncio` had nurseries like `Trio`, we would create a nursery in *Application.run_async*, and run the given coroutine in that nursery.

Not threadsafe.

create_merged_style() → `prompt_toolkit.styles.base.BaseStyle`

Generate a new merged style for the application.

property current_buffer: `prompt_toolkit.buffer.Buffer`

The currently focused `Buffer`.

(This returns a dummy `Buffer` when none of the actual buffers has the focus. In this case, it's really not practical to check for `None` values or catch exceptions every time.)

property current_search_state: `prompt_toolkit.search.SearchState`

Return the current `SearchState`. (The one for the focused `BufferControl`.)

exit(*result: Optional[prompt_toolkit.application.application._AppResult] = None, exception: Optional[Union[BaseException, Type[BaseException]]] = None, style: str = ''*) → `None`

Exit application.

Note: If *Application.exit* is called before *Application.run()* is called, then the *Application* won't exit (because the *Application.future* doesn't correspond to the current run). Use a *pre_run* hook and an event to synchronize the closing if there's a chance this can happen.

Parameters

- **result** – Set this result for the application.
- **exception** – Set this exception as the result for an application. For a prompt, this is often `EOFError` or `KeyboardInterrupt`.
- **style** – Apply this style on the whole content when quitting, often this is 'class:exiting' for a prompt. (Used when *erase_when_done* is not set.)

get_edit_mode() → `prompt_toolkit.enums.EditingMode`

Returns the editing mode enum defined in the configuration.

get_used_style_strings() → `List[str]`

Return a list of used style strings. This is helpful for debugging, and for writing a new *Style*.

invalidate() → `None`

Thread safe way of sending a repaint trigger to the input event loop.

property invalidated: `bool`

True when a redraw operation has been scheduled.

property is_done: `bool`

property is_running: `bool`

True when the application is currently active/running.

key_processor

The *InputProcessor* instance.

classmethod launch() → *None*

Launches the app.

load_container() → *prompt_toolkit.layout.containers.FloatContainer*

Loads the root container for this application.

Returns The root container for this app

load_key_bindings() → *None*

Loads the application's key bindings.

load_output() → *Output*

Creates the output for this application to use.

Returns A prompt-toolkit output instance

property notebook: *Optional[TuiNotebook]*

Return the currently active notebook.

notebook_class: *Type[Notebook]*

open_file(*path: pathlib.Path, read_only: bool = False*) → *None*

Creates a tab for a file.

Parameters

- **path** – The file path of the notebook file to open
- **read_only** – If true, the file should be opened read_only

open_files() → *None*

Opens the files defined in the configuration.

post_load() → *None*

Allows subclasses to define additional loading steps.

pre_run(*app: Optional[prompt_toolkit.application.application.Application] = None*) → *None*

Called during the 'pre-run' stage of application loading.

print_text(*text: Optional[Union[str, MagicFormattedText, List[Union[Tuple[str, str], Tuple[str, str, Callable[[prompt_toolkit.mouse_events.MouseEvent], None]]], Callable[[], Any]]], style: Optional[prompt_toolkit.styles.base.BaseStyle] = None*) → *None*

Print a list of (style_str, text) tuples to the output. (When the UI is running, this method has to be called through *run_in_terminal*, otherwise it will destroy the UI.)

Parameters

- **text** – List of (style_str, text) tuples.
- **style** – Style class to use. Defaults to the active style in the CLI.

quoted_insert

Quoted insert. This flag is set if we go into quoted insert mode.

refresh() → *None*

Reset all tabs.

remove_float(*float_container: Float*) → *None*

Removes a float to the application.

render_counter

Render counter. This one is increased every time the UI is rendered. It can be used as a key for caching certain information during one rendering.

reset() → *None*

Reset everything, for reading the next input.

run(*pre_run: Optional[Callable[[], None]] = None, set_exception_handler: bool = True, handle_sigint: bool = True, in_thread: bool = False*) → *prompt_toolkit.application.application._AppResult*

A blocking ‘run’ call that waits until the UI is finished.

This will start the current asyncio event loop. If no loop is set for the current thread, then it will create a new loop. If a new loop was created, this won’t close the new loop (if *in_thread=False*).

Parameters

- **pre_run** – Optional callable, which is called right after the “reset” of the application.
- **set_exception_handler** – When set, in case of an exception, go out of the alternate screen and hide the application, display the exception, and wait for the user to press ENTER.
- **in_thread** – When true, run the application in a background thread, and block the current thread until the application terminates. This is useful if we need to be sure the application won’t use the current event loop (asyncio does not support nested event loops). A new event loop will be created in this background thread, and that loop will also be closed when the background thread terminates. When this is used, it’s especially important to make sure that all asyncio background tasks are managed through *get_app().create_background_task()*, so that unfinished tasks are properly cancelled before the event loop is closed. This is used for instance in ptython.
- **handle_sigint** – Handle SIGINT signal. Call the key binding for *Keys.SIGINT*. (This only works in the main thread.)

async run_async(*pre_run: Optional[Callable[[], None]] = None, set_exception_handler: bool = True, handle_sigint: bool = True, slow_callback_duration: float = 0.5*) → *prompt_toolkit.application.application._AppResult*

Run the *prompt_toolkit Application* until *exit()* has been called. Return the value that was passed to *exit()*.

This is the main entry point for a *prompt_toolkit Application* and usually the only place where the event loop is actually running.

Parameters

- **pre_run** – Optional callable, which is called right after the “reset” of the application.
- **set_exception_handler** – When set, in case of an exception, go out of the alternate screen and hide the application, display the exception, and wait for the user to press ENTER.
- **handle_sigint** – Handle SIGINT signal if possible. This will call the *<sigint>* key binding when a SIGINT is received. (This only works in the main thread.)
- **slow_callback_duration** – Display warnings if code scheduled in the asyncio event loop takes more time than this. The asyncio default of *0.1* is sometimes not sufficient on a slow system, because exceptionally, the drawing of the app, which happens in the event loop, can take a bit longer from time to time.

async run_system_command(*command: str, wait_for_enter: bool = True, display_before_text: Optional[Union[str, MagicFormattedText, List[Union[Tuple[str, str], Tuple[str, str, Callable[[prompt_toolkit.mouse_events.MouseEvent], None]]], Callable[[[], Any]]] = "", wait_text: str = 'Press ENTER to continue...']*) → *None*

Run system command (While hiding the prompt. When finished, all the output will scroll above the prompt.)

Parameters

- **command** – Shell command to be executed.
- **wait_for_enter** – FWait for the user to press enter, when the command is finished.
- **display_before_text** – If given, text to be displayed before the command executes.

Returns A *Future* object.

set_edit_mode(*mode*: *prompt_toolkit.enums.EditingMode*) → *None*

Sets the keybindings for editing mode.

Parameters **mode** – One of default, vi, or emacs

suspend_to_background(*suspend_group*: *bool = True*) → *None*

(Not thread safe – to be called from inside the key bindings.) Suspend process.

Parameters **suspend_group** – When true, suspend the whole process group. (This is the default, and probably what you want.)

property tab: *Optional[Tab]*

Return the currently selected tab container object.

property tab_idx: *int*

Gets the current tab index.

timeoutlen

Like Vim’s *timeoutlen* option. This can be *None* or a float. For instance, suppose that we have a key binding AB and a second key binding A. If the user presses A and then waits, we don’t handle this binding yet (unless it was marked ‘eager’), because we don’t know what will follow. This timeout is the maximum amount of time that we wait until we call the handlers anyway. Pass *None* to disable this timeout.

ttimeoutlen

When to flush the input (For flushing escape keys.) This is important on terminals that use vt100 input. We can’t distinguish the escape key from for instance the left-arrow key, if we don’t know what follows after “x1b”. This little timer will consider “x1b” to be escape if nothing did follow in this time span. This seems to work like the *ttimeoutlen* option in Vim.

update_style(*query*: *Optional[TerminalQuery] = None*, *pygments_style*: *Optional[str] = None*, *color_scheme*: *Optional[str] = None*) → *None*

Updates the application’s style when the syntax theme is changed.

vi_state

Vi state. (For Vi key bindings.)

euporie.app.current

Defines functions which allow access to typed version of the current application.

Functions

<i>get_base_app()</i>	Get the current application.
<i>get_dump_app()</i>	Get the current application.
<i>get_tui_app()</i>	Get the current application.

euporie.app.current.get_base_app

`euporie.app.current.get_base_app()` → *EuporieApp*
 Get the current application.

euporie.app.current.get_dump_app

`euporie.app.current.get_dump_app()` → *DumpApp*
 Get the current application.

euporie.app.current.get_tui_app

`euporie.app.current.get_tui_app()` → *TuiApp*
 Get the current application.

`euporie.app.current.get_base_app()` → *EuporieApp*
 Get the current application.

`euporie.app.current.get_dump_app()` → *DumpApp*
 Get the current application.

`euporie.app.current.get_tui_app()` → *TuiApp*
 Get the current application.

euporie.app.dump

Concerns dumping output.

Classes

<i>DumpApp</i> (**kwargs)	An application which dumps the layout to the output then exits.
---------------------------	---

euporie.app.dump.DumpApp

class `euporie.app.dump.DumpApp(**kwargs: Any)`
 An application which dumps the layout to the output then exits.

class `euporie.app.dump.DumpApp(**kwargs: Any)`
 Bases: *euporie.app.base.EuporieApp*

An application which dumps the layout to the output then exits.

add_float(*float_container: Float*) → *None*
 Adds a float to the application.

async cancel_and_wait_for_background_tasks() → *None*
 Cancel all background tasks, and wait for the cancellation to be done. If any of the background tasks raised an exception, this will also propagate the exception.

(If we had nurseries like Trio, this would be the `__aexit__` of a nursery.)

property cell: `Optional[InteractiveCell]`

Return the currently active cell.

cleanup_closed_tab(*tab*: `euporie.tab.Tab`) → `None`

Remove a tab container from the current instance of the app.

Parameters *tab* – The closed instance of the tab container

close_tab(*tab*: `Optional[Tab]` = `None`) → `None`

Closes a notebook tab.

Parameters *tab* – The instance of the tab to close. If `None`, the currently selected tab will be closed.

property color_depth: `prompt_toolkit.output.color_depth.ColorDepth`

The active `ColorDepth`.

The current value is determined as follows:

- If a color depth was given explicitly to this application, use that value.
- Otherwise, fall back to the color depth that is reported by the `Output` implementation. If the `Output` class was created using `output.defaults.create_output`, then this value is coming from the `$PROMPT_TOOLKIT_COLOR_DEPTH` environment variable.

command_palette: `Optional[CommandPalette]`

container_statuses: `ContainerStatusDict`

context: `Optional[contextvars.Context]`

cpr_not_supported_callback() → `None`

Called when we don't receive the cursor position response in time.

create_background_task(*coroutine*: `Awaitable[None]`) → `_asyncio.Task`

Start a background task (coroutine) for the running application. When the *Application* terminates, unfinished background tasks will be cancelled.

If `asyncio` had nurseries like `Trio`, we would create a nursery in `Application.run_async`, and run the given coroutine in that nursery.

Not threadsafe.

create_merged_style() → `prompt_toolkit.styles.base.BaseStyle`

Generate a new merged style for the application.

property current_buffer: `prompt_toolkit.buffer.Buffer`

The currently focused `Buffer`.

(This returns a dummy `Buffer` when none of the actual buffers has the focus. In this case, it's really not practical to check for `None` values or catch exceptions every time.)

property current_search_state: `prompt_toolkit.search.SearchState`

Return the current `SearchState`. (The one for the focused `BufferControl`.)

exit(*result*: `Optional[prompt_toolkit.application.application._AppResult]` = `None`, *exception*: `Optional[Union[BaseException, Type[BaseException]]]` = `None`, *style*: `str` = `"`) → `None`
Exit application.

Note: If `Application.exit` is called before `Application.run()` is called, then the *Application* won't exit (because the *Application.future* doesn't correspond to the current run). Use a *pre_run* hook and an event to synchronize the closing if there's a chance this can happen.

Parameters

- **result** – Set this result for the application.
- **exception** – Set this exception as the result for an application. For a prompt, this is often *EOFError* or *KeyboardInterrupt*.
- **style** – Apply this style on the whole content when quitting, often this is ‘class:exiting’ for a prompt. (Used when *erase_when_done* is not set.)

floats: `list[Float]`

full_screen: `bool`

future: `Optional[Future[_AppResult]]`

get_edit_mode() → `prompt_toolkit.enums.EditingMode`
Returns the editing mode enum defined in the configuration.

get_used_style_strings() → `List[str]`
Return a list of used style strings. This is helpful for debugging, and for writing a new *Style*.

invalidate() → `None`
Thread safe way of sending a repaint trigger to the input event loop.

property invalidated: `bool`
True when a redraw operation has been scheduled.

property is_done: `bool`

property is_running: `bool`
True when the application is currently active/running.

key_processor
The *InputProcessor* instance.

classmethod launch() → `None`
Launches the app.

load_container() → `prompt_toolkit.layout.containers.FloatContainer`
Returns a container with all opened tabs.

load_key_bindings() → `None`
Loads the application’s key bindings.

load_output() → `Output`
Loads the output.

Depending on the application configuration, will set the output to a file, to stdout, or to a temporary file so the output can be displayed in a pager.

Returns A container for notebook output

load_tabs() → `Sequence[AnyContainer]`
Returns the currently opened tabs for the printing container.

loop: `Optional[AbstractEventLoop]`

property notebook: `Optional[TuiNotebook]`
Return the currently active notebook.

notebook_class
alias of `euporie.notebook.DumpNotebook`

open_file(*path*: *pathlib.Path*, *read_only*: *bool* = *False*) → *None*

Creates a tab for a file.

Parameters

- **path** – The file path of the notebook file to open
- **read_only** – If true, the file should be opened read_only

open_files() → *None*

Opens the files defined in the configuration.

post_load() → *None*

Allows subclasses to define additional loading steps.

pre_exit(*app*: *Application*) → *None*

Close the app after dumping, optionally piping output to a pager.

pre_run(*app*: *Optional[prompt_toolkit.application.application.Application]* = *None*) → *None*

Called during the ‘pre-run’ stage of application loading.

pre_run_callables: *List[Callable[[], None]]*

print_text(*text*: *Optional[Union[str, MagicFormattedText, List[Union[Tuple[str, str], Tuple[str, str, Callable[[prompt_toolkit.mouse_events.MouseEvent], None]]], Callable[[], Any]]], style: Optional[prompt_toolkit.styles.base.BaseStyle]* = *None*) → *None*

Print a list of (style_str, text) tuples to the output. (When the UI is running, this method has to be called through *run_in_terminal*, otherwise it will destroy the UI.)

Parameters

- **text** – List of (style_str, text) tuples.
- **style** – Style class to use. Defaults to the active style in the CLI.

quoted_insert

Quoted insert. This flag is set if we go into quoted insert mode.

refresh() → *None*

Reset all tabs.

remove_float(*float_container*: *Float*) → *None*

Removes a float to the application.

render_counter

Render counter. This one is increased every time the UI is rendered. It can be used as a key for caching certain information during one rendering.

reset() → *None*

Reset everything, for reading the next input.

run(*pre_run*: *Optional[Callable[[], None]]* = *None*, *set_exception_handler*: *bool* = *True*, *handle_sigint*: *bool* = *True*, *in_thread*: *bool* = *False*) → *prompt_toolkit.application.application._AppResult*

A blocking ‘run’ call that waits until the UI is finished.

This will start the current asyncio event loop. If no loop is set for the current thread, then it will create a new loop. If a new loop was created, this won’t close the new loop (if *in_thread=False*).

Parameters

- **pre_run** – Optional callable, which is called right after the “reset” of the application.
- **set_exception_handler** – When set, in case of an exception, go out of the alternate screen and hide the application, display the exception, and wait for the user to press ENTER.

- **in_thread** – When true, run the application in a background thread, and block the current thread until the application terminates. This is useful if we need to be sure the application won't use the current event loop (asyncio does not support nested event loops). A new event loop will be created in this background thread, and that loop will also be closed when the background thread terminates. When this is used, it's especially important to make sure that all asyncio background tasks are managed through `get_app().create_background_task()`, so that unfinished tasks are properly cancelled before the event loop is closed. This is used for instance in ptython.
- **handle_sigint** – Handle SIGINT signal. Call the key binding for `Keys.SIGINT`. (This only works in the main thread.)

async run_async(*pre_run*: *Optional*[*Callable*[[], *None*]] = *None*, *set_exception_handler*: *bool* = *True*, *handle_sigint*: *bool* = *True*, *slow_callback_duration*: *float* = 0.5) → `prompt_toolkit.application.application._AppResult`

Run the `prompt_toolkit Application` until `exit()` has been called. Return the value that was passed to `exit()`.

This is the main entry point for a `prompt_toolkit Application` and usually the only place where the event loop is actually running.

Parameters

- **pre_run** – Optional callable, which is called right after the “reset” of the application.
- **set_exception_handler** – When set, in case of an exception, go out of the alternate screen and hide the application, display the exception, and wait for the user to press ENTER.
- **handle_sigint** – Handle SIGINT signal if possible. This will call the `<sigint>` key binding when a SIGINT is received. (This only works in the main thread.)
- **slow_callback_duration** – Display warnings if code scheduled in the asyncio event loop takes more time than this. The asyncio default of 0.1 is sometimes not sufficient on a slow system, because exceptionally, the drawing of the app, which happens in the event loop, can take a bit longer from time to time.

async run_system_command(*command*: *str*, *wait_for_enter*: *bool* = *True*, *display_before_text*: *Optional*[*Union*[*str*, *MagicFormattedText*, *List*[*Union*[*Tuple*[*str*, *str*], *Tuple*[*str*, *str*, *Callable*[[*prompt_toolkit.mouse_events.MouseEvent*, *None*]]], *Callable*[[], *Any*]]] = "", *wait_text*: *str* = 'Press ENTER to continue...') → *None*

Run system command (While hiding the prompt. When finished, all the output will scroll above the prompt.)

Parameters

- **command** – Shell command to be executed.
- **wait_for_enter** – FWait for the user to press enter, when the command is finished.
- **display_before_text** – If given, text to be displayed before the command executes.

Returns A *Future* object.

set_edit_mode(*mode*: `prompt_toolkit.enums.EditingMode`) → *None*
Sets the keybindings for editing mode.

Parameters **mode** – One of default, vi, or emacs

suspend_to_background(*suspend_group*: *bool* = *True*) → *None*
(Not thread safe – to be called from inside the key bindings.) Suspend process.

Parameters **suspend_group** – When true, suspend the whole process group. (This is the default, and probably what you want.)

property tab: `Optional[Tab]`

Return the currently selected tab container object.

property tab_idx: `int`

Gets the current tab index.

tabs: `MutableSequence[Tab]`

timeoutlen

Like Vim's *timeoutlen* option. This can be *None* or a float. For instance, suppose that we have a key binding AB and a second key binding A. If the user presses A and then waits, we don't handle this binding yet (unless it was marked 'eager'), because we don't know what will follow. This timeout is the maximum amount of time that we wait until we call the handlers anyway. Pass *None* to disable this timeout.

ttimeoutlen

When to flush the input (For flushing escape keys.) This is important on terminals that use vt100 input. We can't distinguish the escape key from for instance the left-arrow key, if we don't know what follows after "x1b". This little timer will consider "x1b" to be escape if nothing did follow in this time span. This seems to work like the *ttimeoutlen* option in Vim.

update_style(*query: Optional[TerminalQuery] = None, pygments_style: Optional[str] = None, color_scheme: Optional[str] = None*) \rightarrow `None`

Updates the application's style when the syntax theme is changed.

vi_state

Vi state. (For Vi key bindings.)

euporie.app.tui

A text base user interface for euporie.

Classes

TuiApp(**kwargs)

A text user interface euporie application.

euporie.app.tui.TuiApp

class `euporie.app.tui.TuiApp`(**kwargs: Any)

A text user interface euporie application.

class `euporie.app.tui.TuiApp`(**kwargs: Any)

Bases: `euporie.app.base.EuporieApp`

A text user interface euporie application.

add_float(*float_container: Float*) \rightarrow `None`

Adds a float to the application.

ask_file(*default: str = "", validate: bool = True, error: Optional[str] = None, completer: Completer = None*) \rightarrow `None`

Display a dialog asking for file name input.

Parameters

- **default** – The default filename to display in the text entry box
- **validate** – Whether to disallow files which do not exist

- **error** – An optional error message to display below the file name
- **completer** – The completer to use for the input field

ask_new_file() → `None`

Prompts the user to name a file.

ask_open_file() → `None`

Prompts the user to open a file.

async cancel_and_wait_for_background_tasks() → `None`

Cancel all background tasks, and wait for the cancellation to be done. If any of the background tasks raised an exception, this will also propagate the exception.

(If we had nurseries like Trio, this would be the `__aexit__` of a nursery.)

property cell: `Optional[InteractiveCell]`

Return the currently active cell.

cleanup_closed_tab(tab: `euporie.tab.Tab`) → `None`

Remove a tab container from the current instance of the app.

Parameters `tab` – The closed instance of the tab container

clipboard: `Clipboard`

close_tab(tab: `Optional[Tab]` = `None`) → `None`

Closes a notebook tab.

Parameters `tab` – The instance of the tab to close. If `None`, the currently selected tab will be closed.

property color_depth: `prompt_toolkit.output.color_depth.ColorDepth`

The active `ColorDepth`.

The current value is determined as follows:

- If a color depth was given explicitly to this application, use that value.
- Otherwise, fall back to the color depth that is reported by the `Output` implementation. If the `Output` class was created using `output.defaults.create_output`, then this value is coming from the `$PROMPT_TOOLKIT_COLOR_DEPTH` environment variable.

command_palette: `Optional[CommandPalette]`

container_statuses: `ContainerStatusDict`

context: `Optional[contextvars.Context]`

cpr_not_supported_callback() → `None`

Called when we don't receive the cursor position response in time.

create_background_task(coroutine: `Awaitable[None]`) → `_asyncio.Task`

Start a background task (coroutine) for the running application. When the *Application* terminates, unfinished background tasks will be cancelled.

If `asyncio` had nurseries like Trio, we would create a nursery in *Application.run_async*, and run the given coroutine in that nursery.

Not threadsafe.

create_merged_style() → `prompt_toolkit.styles.base.BaseStyle`

Generate a new merged style for the application.

property current_buffer: `prompt_toolkit.buffer.Buffer`

The currently focused Buffer.

(This returns a dummy Buffer when none of the actual buffers has the focus. In this case, it's really not practical to check for *None* values or catch exceptions every time.)

property current_search_state: `prompt_toolkit.search.SearchState`

Return the current SearchState. (The one for the focused BufferControl.)

dialog(*title: AnyFormattedText, body: AnyContainer, buttons: dict[str, Optional[Callable]], to_focus: Optional[AnyContainer] = None*) → *None*

Display a modal dialog above the application.

Returns focus to the previously selected control when closed.

Parameters

- **title** – The title of the dialog. Can be formatted text.
- **body** – The container to use as the main body of the dialog.
- **buttons** – A dictionary mapping text to display as dialog buttons to callbacks to run when the button is clicked. If the callback is *None*, the dialog will be closed without running a callback.
- **to_focus** – The control to focus when the dialog is displayed.

exit(***kwargs: Any*) → *None*

Check for unsaved files before closing.

Creates a chain of close file commands, where the callback for each triggers the closure of the next. The closing process can be cancelled anywhere along the chain.

Parameters ****kwargs** – Unused key word arguments

floats: `list[Float]`

format_status(*part: "Literal['left', 'right']"*) → *StyleAndTextTuples*

Formats the fields in the statusbar generated by the current tab.

Parameters **part** – 'left' to return the fields on the left side of the statusbar, and 'right' to return the fields on the right

Returns A list of style and text tuples for display in the statusbar

format_title() → *StyleAndTextTuples*

Formats the tab's title for display in the top right of the app.

full_screen: `bool`

future: `Optional[Future[_AppResult]]`

get_edit_mode() → `prompt_toolkit.enums.EditingMode`

Returns the editing mode enum defined in the configuration.

get_used_style_strings() → `List[str]`

Return a list of used style strings. This is helpful for debugging, and for writing a new *Style*.

help_about() → *None*

Displays an about dialog.

help_keys() → *None*

Displays details of registered key-bindings in a dialog.

help_logs() → *None*

Displays a dialog with logs.

invalidate() → *None*
Thread safe way of sending a repaint trigger to the input event loop.

property invalidated: *bool*
True when a redraw operation has been scheduled.

property is_done: *bool*

property is_running: *bool*
True when the application is currently active/running.

key_processor
The *InputProcessor* instance.

classmethod launch() → *None*
Launches the app.

load_clipboard() → *None*
Determines which clipboard mechanism to use.

load_container() → *prompt_toolkit.layout.containers.FloatContainer*
Builds the main application layout.

load_key_bindings() → *None*
Loads the application's key bindings.

load_output() → *Output*
Creates the output for this application to use.

Returns A prompt-toolkit output instance

loop: *Optional[AbstractEventLoop]*

menu_container: *MenuContainer*

property notebook: *Optional[TuiNotebook]*
Return the currently active notebook.

notebook_class
alias of *euporie.notebook.TuiNotebook*

open_file(path: *pathlib.Path*, read_only: *bool* = *False*) → *None*
Creates a tab for a file.

Parameters

- **path** – The file path of the notebook file to open
- **read_only** – If true, the file should be opened read_only

open_files() → *None*
Opens the files defined in the configuration.

post_load() → *None*
Continues loading the app.

pre_run(app: *Optional[prompt_toolkit.application.application.Application]* = *None*) → *None*
Called during the 'pre-run' stage of application loading.

pre_run_callables: *List[Callable[[], None]]*

print_text(text: *Optional[Union[str, MagicFormattedText, List[Union[Tuple[str, str], Tuple[str, str, Callable[[prompt_toolkit.mouse_events.MouseEvent], None]]], Callable[[], Any]]]*, style: *Optional[prompt_toolkit.styles.base.BaseStyle] = None*) → None

Print a list of (style_str, text) tuples to the output. (When the UI is running, this method has to be called through `run_in_terminal`, otherwise it will destroy the UI.)

Parameters

- **text** – List of (style_str, text) tuples.
- **style** – Style class to use. Defaults to the active style in the CLI.

quoted_insert

Quoted insert. This flag is set if we go into quoted insert mode.

refresh() → None

Reset all tabs.

remove_float(float_container: Float) → None

Removes a float to the application.

render_counter

Render counter. This one is increased every time the UI is rendered. It can be used as a key for caching certain information during one rendering.

reset() → None

Reset everything, for reading the next input.

run(pre_run: *Optional[Callable[[], None]] = None*, set_exception_handler: *bool = True*, handle_sigint: *bool = True*, in_thread: *bool = False*) → prompt_toolkit.application.application._AppResult

A blocking ‘run’ call that waits until the UI is finished.

This will start the current asyncio event loop. If no loop is set for the current thread, then it will create a new loop. If a new loop was created, this won’t close the new loop (if `in_thread=False`).

Parameters

- **pre_run** – Optional callable, which is called right after the “reset” of the application.
- **set_exception_handler** – When set, in case of an exception, go out of the alternate screen and hide the application, display the exception, and wait for the user to press ENTER.
- **in_thread** – When true, run the application in a background thread, and block the current thread until the application terminates. This is useful if we need to be sure the application won’t use the current event loop (asyncio does not support nested event loops). A new event loop will be created in this background thread, and that loop will also be closed when the background thread terminates. When this is used, it’s especially important to make sure that all asyncio background tasks are managed through `get_app().create_background_task()`, so that unfinished tasks are properly cancelled before the event loop is closed. This is used for instance in ptpython.
- **handle_sigint** – Handle SIGINT signal. Call the key binding for `Keys.SIGINT`. (This only works in the main thread.)

async run_async(pre_run: *Optional[Callable[[], None]] = None*, set_exception_handler: *bool = True*, handle_sigint: *bool = True*, slow_callback_duration: *float = 0.5*) → prompt_toolkit.application.application._AppResult

Run the prompt_toolkit `Application` until `exit()` has been called. Return the value that was passed to `exit()`.

This is the main entry point for a prompt_toolkit `Application` and usually the only place where the event loop is actually running.

Parameters

- **pre_run** – Optional callable, which is called right after the “reset” of the application.
- **set_exception_handler** – When set, in case of an exception, go out of the alternate screen and hide the application, display the exception, and wait for the user to press ENTER.
- **handle_sigint** – Handle SIGINT signal if possible. This will call the *<sigint>* key binding when a SIGINT is received. (This only works in the main thread.)
- **slow_callback_duration** – Display warnings if code scheduled in the asyncio event loop takes more time than this. The asyncio default of *0.1* is sometimes not sufficient on a slow system, because exceptionally, the drawing of the app, which happens in the event loop, can take a bit longer from time to time.

async_run_system_command(*command: str*, *wait_for_enter: bool = True*, *display_before_text: Optional[Union[str, MagicFormattedText, List[Union[Tuple[str, str], Tuple[str, str, Callable[[prompt_toolkit.mouse_events.MouseEvent], None]]], Callable[[[], Any]] = "], wait_text: str = 'Press ENTER to continue...']*) → *None*

Run system command (While hiding the prompt. When finished, all the output will scroll above the prompt.)

Parameters

- **command** – Shell command to be executed.
- **wait_for_enter** – FWait for the user to press enter, when the command is finished.
- **display_before_text** – If given, text to be displayed before the command executes.

Returns A *Future* object.

set_edit_mode(*mode: prompt_toolkit.enums.EditingMode*) → *None*

Sets the keybindings for editing mode.

Parameters **mode** – One of default, vi, or emacs

suspend_to_background(*suspend_group: bool = True*) → *None*

(Not thread safe – to be called from inside the key bindings.) Suspend process.

Parameters **suspend_group** – When true, suspend the whole process group. (This is the default, and probably what you want.)

property tab: *Optional[Tab]*

Return the currently selected tab container object.

tab_container() → *AnyContainer*

Returns a container with all opened tabs.

Returns A vertical split containing the opened tab containers.

property tab_idx: *int*

Gets the current tab index.

tabs: *MutableSequence[Tab]*

timeoutlen

Like Vim’s *timeoutlen* option. This can be *None* or a float. For instance, suppose that we have a key binding AB and a second key binding A. If the user presses A and then waits, we don’t handle this binding yet (unless it was marked ‘eager’), because we don’t know what will follow. This timeout is the maximum amount of time that we wait until we call the handlers anyway. Pass *None* to disable this timeout.

timeoutlen

When to flush the input (For flushing escape keys.) This is important on terminals that use vt100 input. We can't distinguish the escape key from for instance the left-arrow key, if we don't know what follows after "x1b". This little timer will consider "x1b" to be escape if nothing did follow in this time span. This seems to work like the *timeoutlen* option in Vim.

update_style(*query*: *Optional*[*TerminalQuery*] = *None*, *pygments_style*: *Optional*[*str*] = *None*,
color_scheme: *Optional*[*str*] = *None*) → *None*

Updates the application's style when the syntax theme is changed.

vi_state

Vi state. (For Vi key bindings.)

5.8.2 euporie.box

Define box border constants.

Classes

<i>BorderLine</i> ([char, width, height, collapse, ...])	Draws a horizontal or vertical line.
<i>Pattern</i> ()	Fill an area with a repeating background pattern.
<i>RoundBorder</i> ()	Box drawing characters with rounded corners.
<i>SquareBorder</i> ()	Box drawing characters, including characters for splits.
<i>ThickVerticalBorder</i> ()	Box drawing characters with thick verticals corners.

euporie.box.BorderLine

class euporie.box.**BorderLine**(*char*: *Optional*[*str*] = *None*, *width*: *Optional*[*int*] = *None*, *height*:
Optional[*int*] = *None*, *collapse*: *bool* = *False*, *style*: *str* = 'class:border-line')

Draws a horizontal or vertical line.

euporie.box.Pattern

class euporie.box.**Pattern**

Fill an area with a repeating background pattern.

euporie.box.RoundBorder

class euporie.box.RoundBorder

Box drawing characters with rounded corners.

euporie.box.SquareBorder

class euporie.box.SquareBorder

Box drawing characters, including characters for splits.

euporie.box.ThickVerticalBorder

class euporie.box.ThickVerticalBorder

Box drawing characters with thick verticals corners.

class euporie.box.BorderLine(char: *Optional[str]* = None, width: *Optional[int]* = None, height: *Optional[int]* = None, collapse: *bool* = False, style: *str* = 'class:border-line')

Bases: `prompt_toolkit.layout.containers.Container`

Draws a horizontal or vertical line.

get_children() → *list*

Return an empty list of the container's children.

get_key_bindings() → *Optional[prompt_toolkit.key_binding.key_bindings.KeyBindingsBase]*

Returns a `KeyBindings` object. These bindings become active when any user control in this container has the focus, except if any containers between this container and the focused user control is modal.

is_modal() → *bool*

When this container is modal, key bindings from parent containers are not taken into account if a user control in this container is focused.

preferred_height(width: *int*, max_available_height: *int*) → *prompt_toolkit.layout.dimension.Dimension*

Return the preferred height of the line.

preferred_width(max_available_width: *int*) → *prompt_toolkit.layout.dimension.Dimension*

Return the preferred width of the line.

reset() → *None*

Resets the state of the line. Does nothing.

write_to_screen(screen: *prompt_toolkit.layout.screen.Screen*, mouse_handlers: *prompt_toolkit.layout.mouse_handlers.MouseHandlers*, write_position: *prompt_toolkit.layout.screen.WritePosition*, parent_style: *str*, erase_bg: *bool*, z_index: *Optional[int]*) → *None*

Draws a continuous line in the `write_position` area.

Parameters

- **screen** – The `Screen` class to which the output has to be written.
- **mouse_handlers** – `prompt_toolkit.layout.mouse_handlers.MouseHandlers`.
- **write_position** – A `prompt_toolkit.layout.screen.WritePosition` object defining where this container should be drawn.
- **erase_bg** – If true, the background will be erased prior to drawing.

- **parent_style** – Style string to pass to the Window object. This will be applied to all content of the windows. `VSplit` and `prompt_toolkit.layout.containers.HSplit` can use it to pass their style down to the windows that they contain.
- **z_index** – Used for propagating `z_index` from parent to child.

class euporie.box.**Pattern**

Bases: `prompt_toolkit.layout.containers.Container`

Fill an area with a repeating background pattern.

get_children() → `list`

Return an empty list of the container's children.

get_key_bindings() → `Optional[prompt_toolkit.key_binding.key_bindings.KeyBindingsBase]`

Returns a `KeyBindings` object. These bindings become active when any user control in this container has the focus, except if any containers between this container and the focused user control is modal.

is_modal() → `bool`

When this container is modal, key bindings from parent containers are not taken into account if a user control in this container is focused.

preferred_height(*width: int, max_available_height: int*) → `prompt_toolkit.layout.dimension.Dimension`

Return an empty dimension (expand to available height).

preferred_width(*max_available_width: int*) → `prompt_toolkit.layout.dimension.Dimension`

Return an empty dimension (expand to available width).

reset() → `None`

Resets the pattern. Does nothing.

write_to_screen(*screen: prompt_toolkit.layout.screen.Screen, mouse_handlers: prompt_toolkit.layout.mouse_handlers.MouseHandlers, write_position: prompt_toolkit.layout.screen.WritePosition, parent_style: str, erase_bg: bool, z_index: Optional[int]*) → `None`

Fill the whole area of `write_position` with a pattern.

Parameters

- **screen** – The `Screen` class to which the output has to be written.
- **mouse_handlers** – `prompt_toolkit.layout.mouse_handlers.MouseHandlers`.
- **write_position** – A `prompt_toolkit.layout.screen.WritePosition` object defining where this container should be drawn.
- **erase_bg** – If true, the background will be erased prior to drawing.
- **parent_style** – Style string to pass to the Window object. This will be applied to all content of the windows. `VSplit` and `prompt_toolkit.layout.containers.HSplit` can use it to pass their style down to the windows that they contain.
- **z_index** – Used for propagating `z_index` from parent to child.

class euporie.box.**RoundBorder**

Bases: `euporie.box.SquareBorder`

Box drawing characters with rounded corners.

BOTTOM_LEFT = ''

BOTTOM_RIGHT = ''

CROSS = ''

```

HORIZONTAL = '-'
INNER_VERTICAL = '|'
NONE = ' '
SPLIT_BOTTOM = ''
SPLIT_LEFT = '┌'
SPLIT_RIGHT = ''
SPLIT_TOP = ''
TOP_LEFT = ''
TOP_RIGHT = ''
VERTICAL = '|'

class euporie.box.SquareBorder
    Bases: prompt_toolkit.widgets.base.Border
    Box drawing characters, including characters for splits.
    BOTTOM_LEFT = '└'
    BOTTOM_RIGHT = ''
    CROSS = ''
    HORIZONTAL = '-'
    INNER_VERTICAL = '|'
    NONE = ' '
    SPLIT_BOTTOM = ''
    SPLIT_LEFT = '┌'
    SPLIT_RIGHT = ''
    SPLIT_TOP = ''
    TOP_LEFT = ''
    TOP_RIGHT = ''
    VERTICAL = '|'

```

5.8.3 euporie.cell

Defines a cell object with input and rich outputs, and related objects.

Functions

<code>get_cell_id(cell_json)</code>	Returns the cell ID field defined in a cell JSON object.
-------------------------------------	--

euporie.cell.get_cell_id

`euporie.cell.get_cell_id(cell_json: dict) → str`

Returns the cell ID field defined in a cell JSON object.

If no cell ID is defined (as per ``:mod:`nbformat`<4.5`), then one is generated and added to the cell.

Parameters `cell_json` – The cell's JSON object as a python dictionary

Returns The ID string

Classes

<code>Cell(index, json, notebook)</code>	A notebook cell element.
<code>CellInputTextArea(cell, *args, **kwargs)</code>	A customized text area for the cell input.
<code>CellStdinTextArea(*args, **kwargs)</code>	A modal text area for user input.
<code>ClickArea(target, text, style)</code>	Any empty widget which focuses <i>target</i> when clicked.
<code>InteractiveCell(index, json, notebook)</code>	An interactive notebook cell.
<code>PagerState(code, cursor_pos, response)</code>	

euporie.cell.Cell

class `euporie.cell.Cell(index: int, json: dict, notebook: Notebook)`

A notebook cell element.

Contains a transparent clickable overlay, which is not displayed when the cell is focused.

euporie.cell.CellInputTextArea

class `euporie.cell.CellInputTextArea(cell: Cell, *args: Any, **kwargs: Any)`

A customized text area for the cell input.

euporie.cell.CellStdinTextArea

class `euporie.cell.CellStdinTextArea(*args: Any, **kwargs: Any)`

A modal text area for user input.

euporie.cell.ClickArea

```
class euporie.cell.ClickArea(target: FocusableElement, text: AnyFormattedText, style: Union[str,
                                Callable[[], str]])
```

Any empty widget which focuses *target* when clicked.

Designed to be used as an overlay for clickable widgets in a FloatContainer.

euporie.cell.InteractiveCell

```
class euporie.cell.InteractiveCell(index: int, json: dict, notebook: TuiNotebook)
```

An interactive notebook cell.

euporie.cell.PagerState

```
class euporie.cell.PagerState(code, cursor_pos, response)
```

```
class euporie.cell.Cell(index: int, json: dict, notebook: Notebook)
```

Bases: `object`

A notebook cell element.

Contains a transparent clickable overlay, which is not displayed when the cell is focused.

```
border_char(name: str) → Callable[..., str]
```

Returns a function which returns the cell border character to display.

```
border_style() → str
```

Determines the style of the cell borders, based on the cell state.

```
property cell_type: str
```

Determine the current cell type.

```
clear_output() → None
```

Remove all outputs from the cell.

```
property execution_count: str
```

Retrieve the execution count from the cell's JSON.

```
exit_edit_mode() → None
```

Removes a cell from edit mode.

```
property focused: bool
```

Determine if the cell currently has focus.

```
property id: str
```

Returns the cell's ID as per the cell JSON.

```
property input: str
```

Fetch the cell's contents from the cell's JSON.

```
inspect() → None
```

Get contextual help for the current cursor position.

```
property language: str
```

Returns the cell's code language.

```
on_output() → None
```

Runs when a message for this cell is received from the kernel.

property outputs: `list[dict[str, Any]]`

Retrieve a list of cell outputs from the cell's JSON.

property prompt: `str`

Determine what should be displayed in the prompt of the cell.

ran(*cell_json: Optional[dict] = None*) → `None`

Callback which runs when the cell has finished running.

reformat() → `None`

Reformats the cell's input.

remove_output_graphic_floats() → `None`

Unregisters the cell's output's graphic floats with the applications.

render_outputs() → `list[prompt_toolkit.layout.containers.Container]`

Generates a list of rendered outputs.

run_or_render(*buffer: Optional[Buffer] = None, wait: bool = False*) → `bool`

Placeholder function for running the cell.

Parameters

- **buffer** – Unused parameter, required when accepting the contents of a cell's input buffer
- **wait** – Has no effect

Returns Always returns True

property selected: `bool`

Determine if the cell currently is selected.

set_cell_type(*cell_type: "Literal['markdown','code','raw']"*, *clear: bool = False*) → `None`

Convert the cell to a different cell type.

Parameters

- **cell_type** – The desired cell type.
- **clear** – If True, cell outputs will be cleared

trigger_refresh() → `None`

Request that the cell to be re-rendered next time it is drawn.

class `euporie.cell.CellInputTextArea`(*cell: Cell, *args: Any, **kwargs: Any*)

Bases: `prompt_toolkit.widgets.base.TextArea`

A customized text area for the cell input.

property accept_handler: `Optional[Callable[[prompt_toolkit.buffer.Buffer], bool]]`

The accept handler. Called when the user accepts the input.

property document: `prompt_toolkit.document.Document`

The *Buffer* document (text + cursor position).

on_cursor_position_changed(*buf: Buffer*) → `None`

Respond to cursor movements.

on_text_changed(*buf: Buffer*) → `None`

Update cell json when the input buffer has been edited.

property text: `str`

The *Buffer* text.

```

class euporie.cell.CellStdinTextArea(*args: Any, **kwargs: Any)
    Bases: prompt_toolkit.widgets.base.TextArea

    A modal text area for user input.

    property accept_handler: Optional[Callable[[prompt_toolkit.buffer.Buffer], bool]]
        The accept handler. Called when the user accepts the input.

    property document: prompt_toolkit.document.Document
        The Buffer document (text + cursor position).

    is_modal() → bool
        Returns true, so the input is always modal.

    property text: str
        The Buffer text.

class euporie.cell.ClickArea(target: FocusableElement, text: AnyFormattedText, style: Union[str,
    Callable[[], str]])
    Bases: object

    Any empty widget which focuses target when clicked.

    Designed to be used as an overlay for clickable widgets in a FloatContainer.

class euporie.cell.InteractiveCell(index: int, json: dict, notebook: TuiNotebook)
    Bases: euporie.cell.Cell

    An interactive notebook cell.

    border_char(name: str) → Callable[..., str]
        Returns a function which returns the cell border character to display.

    border_style() → str
        Determines the style of the cell borders, based on the cell state.

    property cell_type: str
        Determine the current cell type.

    clear_output() → None
        Remove all outputs from the cell.

    async edit_in_editor() → None
        Edit the cell in $EDITOR.

    enter_edit_mode() → None
        Set the cell to edit mode.

    property execution_count: str
        Retrieve the execution count from the cell's JSON.

    exit_edit_mode() → None
        Removes a cell from edit mode.

    property focused: bool
        Determine if the cell currently has focus.

    get_input(send: Callable[[str], Any], prompt: str = 'Please enter a value:', password: bool = False) →
        None
        Prompts the user for input and sends the result to the kernel.

    property id: str
        Returns the cell's ID as per the cell JSON.

```

property input: `str`

Fetch the cell's contents from the cell's JSON.

inspect() → `None`

Get contextual help for the current cursor position.

property language: `str`

Returns the cell's code language.

on_output() → `None`

Runs when a message for this cell is received from the kernel.

property outputs: `list[dict[str, Any]]`

Retrieve a list of cell outputs from the cell's JSON.

property prompt: `str`

Determine what should be displayed in the prompt of the cell.

ran(*cell_json: Optional[dict] = None*) → `None`

Callback which runs when the cell has finished running.

reformat() → `None`

Reformats the cell's input.

remove_output_graphic_floats() → `None`

Unregisters the cell's output's graphic floats with the applications.

render_outputs() → `list[prompt_toolkit.layout.containers.Container]`

Generates a list of rendered outputs.

run_or_render(*buffer: Optional[Buffer] = None, wait: bool = False*) → `bool`

Run code cells, or render markdown cells, optionally advancing.

Parameters

- **buffer** – Unused parameter, required when accepting the contents of a cell's input buffer
- **wait** – If True, block the main thread until the cell has finished running

Returns Always return True

select() → `None`

Selects this cell.

property selected: `bool`

Determine if the cell currently is selected.

set_cell_type(*cell_type: "Literal['markdown','code','raw']"*, *clear: bool = False*) → `None`

Convert the cell to a different cell type.

Parameters

- **cell_type** – The desired cell type.
- **clear** – If True, cell outputs will be cleared

split() → `None`

Split the cell at the current cursor position.

trigger_refresh() → `None`

Request that the cell to be re-rendered next time it is drawn.

`euporie.cell.get_cell_id`(*cell_json: dict*) → `str`

Returns the cell ID field defined in a cell JSON object.

If no cell ID is defined (as per ``:mod:`nbformat`<4.5`), then one is generated and added to the cell.

Parameters `cell_json` – The cell’s JSON object as a python dictionary

Returns The ID string

5.8.4 euporie.commands

Defines a centralized command system.

Modules

<code>euporie.commands.base</code>	Defines a command object for use in key-bindings, menus, and the command palette.
<code>euporie.commands.buffer</code>	Defines commands relating to editing buffers.
<code>euporie.commands.cell</code>	Defines commands relating to cells.
<code>euporie.commands.cell_output</code>	Defines commands relating to cell outputs.
<code>euporie.commands.completions</code>	Defines commands relating to completions.
<code>euporie.commands.config</code>	Define commands at the application level.
<code>euporie.commands.format</code>	Define commands at the application level.
<code>euporie.commands.notebook</code>	Defines commands relating to notebooks.
<code>euporie.commands.pager</code>	Defines commands for the pager.
<code>euporie.commands.registry</code>	Defines functions to manage the command registry.
<code>euporie.commands.search</code>	Defines commands related to searching.
<code>euporie.commands.suggestions</code>	Defines command relating to suggestions.
<code>euporie.commands.tui</code>	Define commands at the application level.

euporie.commands.base

Defines a command object for use in key-bindings, menus, and the command palette.

Functions

<code>add(**kwargs)</code>	Adds a command to the centralized command system.
<code>get(name)</code>	Get a command from the centralized command system by name.

euporie.commands.base.add

`euporie.commands.base.add(**kwargs: Any) → Callable`

Adds a command to the centralized command system.

euporie.commands.base.get

`euporie.commands.base.get(name: str) → euporie.commands.base.Command`

Get a command from the centralized command system by name.

Parameters **name** – The name of the command to retrieve

Returns The requested command object

Raises **KeyError** – Raised if the named command is not found

Classes

<code>Command(handler, Optional[Awaitable[Any]], ...)</code>	Wraps a function so it can be used as a key-binding or a menu item.
--	---

euporie.commands.base.Command

```
class euporie.commands.base.Command(handler: Callable[..., Optional[Awaitable[Any]]], *, filter:
    FilterOrBool = True, hidden: FilterOrBool = False, name:
    Optional[str] = None, title: Optional[str] = None, menu_title:
    Optional[str] = None, description: Optional[str] = None, group:
    Optional[str] = None, toggled: Optional[Filter] = None, keys:
    Optional[AnyKeys] = None, eager: FilterOrBool = False, is_global:
    FilterOrBool = False, save_before: Callable[[KeyPressEvent], bool]
    = <function Command.<lambda>>, record_in_macro: FilterOrBool
    = True)
```

Wraps a function so it can be used as a key-binding or a menu item.

```
class euporie.commands.base.Command(handler: Callable[..., Optional[Awaitable[Any]]], *, filter:
    FilterOrBool = True, hidden: FilterOrBool = False, name:
    Optional[str] = None, title: Optional[str] = None, menu_title:
    Optional[str] = None, description: Optional[str] = None, group:
    Optional[str] = None, toggled: Optional[Filter] = None, keys:
    Optional[AnyKeys] = None, eager: FilterOrBool = False, is_global:
    FilterOrBool = False, save_before: Callable[[KeyPressEvent], bool]
    = <function Command.<lambda>>, record_in_macro: FilterOrBool
    = True)
```

Bases: `object`

Wraps a function so it can be used as a key-binding or a menu item.

add_keys(keys: Optional[AnyKeys]) → `Command`

Adds keyboard shortcuts to the current command.

bind(key_bindings: KeyBindingsBase, keys: Optional[AnyKeys] = None) → `None`

Add the current commands to a set of key bindings.

Parameters

- **key_bindings** – The set of key bindings to bind to
- **keys** – Additional keys to bind to the command

property key_bindings: `Sequence[Binding]`

Returns a list of key-bindings given to the current command.

property key_handler: `KeyHandlerCallable`

Returns a key handler for the command.

property menu: `euporie.menu.item.MenuItem`

Returns a menu item for the command.

property menu_handler: `Callable[[], None]`

Returns a menu handler for the command.

run() `→ None`

Runs the command's handler.

`euporie.commands.base.add(**kwargs: Any) → Callable`

Adds a command to the centralized command system.

`euporie.commands.base.get(name: str) → euporie.commands.base.Command`

Get a command from the centralized command system by name.

Parameters `name` – The name of the command to retrieve

Returns The requested command object

Raises `KeyError` – Raised if the named command is not found

euporie.commands.buffer

Defines commands relating to editing buffers.

Functions

<code>cancel_selection(event)</code>	Cancel the selection.
<code>copy_selection()</code>	Adds the current selection to the clipboard.
<code>cut_line()</code>	Removes the current line adds it to the clipboard.
<code>cut_selection()</code>	Removes the current selection and adds it to the clipboard.
<code>delete_selection()</code>	Delete the contents of the current selection.
<code>dent_buffer(event[, indenting])</code>	Indent or unindent the current or selected lines in a buffer.
<code>duplicate_line()</code>	Duplicate the current line.
<code>duplicate_selection()</code>	Duplicate the current line.
<code>end_macro()</code>	Stop recording a macro.
<code>extend_selection(event)</code>	Extend the selection.
<code>go_to_end_of_line()</code>	Move the cursor to the end of the line.
<code>go_to_end_of_paragraph()</code>	Move the cursor to the end of the current paragraph.
<code>go_to_start_of_line()</code>	Move the cursor to the start of the line.
<code>go_to_start_of_paragraph()</code>	Move the cursor to the start of the current paragraph.
<code>if_no_repeat(event)</code>	Returns True when the previous event was delivered to another handler.
<code>indent_lines(event)</code>	Inndent the current or selected lines.
<code>move_cursor_left()</code>	Move back a character, or up a line.
<code>move_cursor_right()</code>	Move forward a character, or down a line.
<code>move_line(n)</code>	Moves the current or selected lines up or down by one or more lines.
<code>move_lines_down()</code>	Move the current or selected lines down by one line.

continues on next page

Table 15 – continued from previous page

<code>move_lines_up()</code>	Move the current or selected lines up by one line.
<code>newline(event)</code>	Insert a new line, replacing any selection and indenting if appropriate.
<code>paste_clipboard()</code>	Paste the clipboard contents, replacing any current selection.
<code>redo()</code>	Redo the last edit.
<code>replace_selection(event)</code>	Replace selection by what is typed.
<code>run_macro()</code>	Re-execute the last keyboard macro defined.
<code>select_all()</code>	Select all text.
<code>show_contextual_help(event)</code>	Displays contextual help.
<code>start_macro()</code>	Start recording a macro.
<code>start_selection(event)</code>	Start a new selection.
<code>toggle_case()</code>	Toggle the case of the current word or selection.
<code>toggle_comment()</code>	Comments or uncomments the current or selected lines.
<code>toggle_overwrite_mode()</code>	Toggle overwrite when using micro editing mode.
<code>type_key(event)</code>	Enter a key.
<code>undo()</code>	Undo the last edit.
<code>unindent_lines(event)</code>	Unindent the current or selected lines.
<code>unshift_move(event)</code>	Used for the shift selection mode.
<code>wrap_selection_cmd(left, right)</code>	Adds strings to either end of the current selection.

euporie.commands.buffer.cancel_selection

`euporie.commands.buffer.cancel_selection(event: KeyPressEvent) → None`
Cancel the selection.

euporie.commands.buffer.copy_selection

`euporie.commands.buffer.copy_selection() → None`
Adds the current selection to the clipboard.

euporie.commands.buffer.cut_line

`euporie.commands.buffer.cut_line() → None`
Removes the current line adds it to the clipboard.

euporie.commands.buffer.cut_selection

`euporie.commands.buffer.cut_selection() → None`
Removes the current selection and adds it to the clipboard.

euporie.commands.buffer.delete_selection

`euporie.commands.buffer.delete_selection()` → `None`

Delete the contents of the current selection.

euporie.commands.buffer.dent_buffer

`euporie.commands.buffer.dent_buffer(event: KeyPressEvent, indenting: bool = True)` → `None`

Indent or unindent the current or selected lines in a buffer.

euporie.commands.buffer.duplicate_line

`euporie.commands.buffer.duplicate_line()` → `None`

Duplicate the current line.

euporie.commands.buffer.duplicate_selection

`euporie.commands.buffer.duplicate_selection()` → `None`

Duplicate the current line.

euporie.commands.buffer.end_macro

`euporie.commands.buffer.end_macro()` → `None`

Stop recording a macro.

euporie.commands.buffer.extend_selection

`euporie.commands.buffer.extend_selection(event: KeyPressEvent)` → `None`

Extend the selection.

euporie.commands.buffer.go_to_end_of_line

`euporie.commands.buffer.go_to_end_of_line()` → `None`

Move the cursor to the end of the line.

euporie.commands.buffer.go_to_end_of_paragraph

`euporie.commands.buffer.go_to_end_of_paragraph()` → `None`

Move the cursor to the end of the current paragraph.

euporie.commands.buffer.go_to_start_of_line

`euporie.commands.buffer.go_to_start_of_line()` → `None`
Move the cursor to the start of the line.

euporie.commands.buffer.go_to_start_of_paragraph

`euporie.commands.buffer.go_to_start_of_paragraph()` → `None`
Move the cursor to the start of the current paragraph.

euporie.commands.buffer.if_no_repeat

`euporie.commands.buffer.if_no_repeat(event: KeyPressEvent)` → `bool`
Returns True when the previous event was delivered to another handler.

euporie.commands.buffer.indent_lines

`euporie.commands.buffer.indent_lines(event: KeyPressEvent)` → `None`
Inndent the current or selected lines.

euporie.commands.buffer.move_cursor_left

`euporie.commands.buffer.move_cursor_left()` → `None`
Move back a character, or up a line.

euporie.commands.buffer.move_cursor_right

`euporie.commands.buffer.move_cursor_right()` → `None`
Move forward a character, or down a line.

euporie.commands.buffer.move_line

`euporie.commands.buffer.move_line(n: int)` → `None`
Moves the current or selected lines up or down by one or more lines.

euporie.commands.buffer.move_lines_down

`euporie.commands.buffer.move_lines_down()` → `None`
Move the current or selected lines down by one line.

euporie.commands.buffer.move_lines_up

`euporie.commands.buffer.move_lines_up()` → `None`

Move the current or selected lines up by one line.

euporie.commands.buffer.newline

`euporie.commands.buffer.newline(event: KeyPressEvent)` → `None`

Insert a new line, replacing any selection and indenting if appropriate.

euporie.commands.buffer.paste_clipboard

`euporie.commands.buffer.paste_clipboard()` → `None`

Paste the clipboard contents, replacing any current selection.

euporie.commands.buffer.redo

`euporie.commands.buffer.redo()` → `None`

Redo the last edit.

euporie.commands.buffer.replace_selection

`euporie.commands.buffer.replace_selection(event: KeyPressEvent)` → `None`

Replace selection by what is typed.

euporie.commands.buffer.run_macro

`euporie.commands.buffer.run_macro()` → `None`

Re-execute the last keyboard macro defined.

euporie.commands.buffer.select_all

`euporie.commands.buffer.select_all()` → `None`

Select all text.

euporie.commands.buffer.show_contextual_help

`euporie.commands.buffer.show_contextual_help(event: KeyPressEvent)` → `None`

Displays contextual help.

euporie.commands.buffer.start_macro

`euporie.commands.buffer.start_macro()` → `None`
Start recording a macro.

euporie.commands.buffer.start_selection

`euporie.commands.buffer.start_selection(event: KeyPressEvent)` → `None`
Start a new selection.

euporie.commands.buffer.toggle_case

`euporie.commands.buffer.toggle_case()` → `None`
Toggle the case of the current word or selection.

euporie.commands.buffer.toggle_comment

`euporie.commands.buffer.toggle_comment()` → `None`
Comments or uncomments the current or selected lines.

euporie.commands.buffer.toggle_overwrite_mode

`euporie.commands.buffer.toggle_overwrite_mode()` → `None`
Toggle overwrite when using micro editing mode.

euporie.commands.buffer.type_key

`euporie.commands.buffer.type_key(event: KeyPressEvent)` → `None`
Enter a key.

euporie.commands.buffer.undo

`euporie.commands.buffer.undo()` → `None`
Undo the last edit.

euporie.commands.buffer.unindent_lines

`euporie.commands.buffer.unindent_lines(event: KeyPressEvent)` → `None`
Unindent the current or selected lines.

euporie.commands.buffer.unshift_move

`euporie.commands.buffer.unshift_move(event: KeyPressEvent) → None`

Used for the shift selection mode.

When called with a shift + movement key press event, moves the cursor as if shift is not pressed.

Parameters **event** – The key press event to process

euporie.commands.buffer.wrap_selection_cmd

`euporie.commands.buffer.wrap_selection_cmd(left: str, right: str) → None`

Adds strings to either end of the current selection.

`euporie.commands.buffer.cancel_selection(event: KeyPressEvent) → None`

Cancel the selection.

`euporie.commands.buffer.copy_selection() → None`

Adds the current selection to the clipboard.

`euporie.commands.buffer.cut_line() → None`

Removes the current line adds it to the clipboard.

`euporie.commands.buffer.cut_selection() → None`

Removes the current selection and adds it to the clipboard.

`euporie.commands.buffer.delete_selection() → None`

Delete the contents of the current selection.

`euporie.commands.buffer.dent_buffer(event: KeyPressEvent, indenting: bool = True) → None`

Indent or unindent the current or selected lines in a buffer.

`euporie.commands.buffer.duplicate_line() → None`

Duplicate the current line.

`euporie.commands.buffer.duplicate_selection() → None`

Duplicate the current line.

`euporie.commands.buffer.end_macro() → None`

Stop recording a macro.

`euporie.commands.buffer.extend_selection(event: KeyPressEvent) → None`

Extend the selection.

`euporie.commands.buffer.go_to_end_of_line() → None`

Move the cursor to the end of the line.

`euporie.commands.buffer.go_to_end_of_paragraph() → None`

Move the cursor to the end of the current paragraph.

`euporie.commands.buffer.go_to_start_of_line() → None`

Move the cursor to the start of the line.

`euporie.commands.buffer.go_to_start_of_paragraph() → None`

Move the cursor to the start of the current paragraph.

`euporie.commands.buffer.if_no_repeat(event: KeyPressEvent) → bool`

Returns True when the previous event was delivered to another handler.

`euporie.commands.buffer.indent_lines(event: KeyPressEvent) → None`

Inndent the current or selected lines.

`euporie.commands.buffer.move_cursor_left()` → `None`
Move back a character, or up a line.

`euporie.commands.buffer.move_cursor_right()` → `None`
Move forward a character, or down a line.

`euporie.commands.buffer.move_line(n: int)` → `None`
Moves the current or selected lines up or down by one or more lines.

`euporie.commands.buffer.move_lines_down()` → `None`
Move the current or selected lines down by one line.

`euporie.commands.buffer.move_lines_up()` → `None`
Move the current or selected lines up by one line.

`euporie.commands.buffer.newline(event: KeyPressEvent)` → `None`
Insert a new line, replacing any selection and indenting if appropriate.

`euporie.commands.buffer.paste_clipboard()` → `None`
Paste the clipboard contents, replacing any current selection.

`euporie.commands.buffer.redo()` → `None`
Redo the last edit.

`euporie.commands.buffer.replace_selection(event: KeyPressEvent)` → `None`
Replace selection by what is typed.

`euporie.commands.buffer.run_macro()` → `None`
Re-execute the last keyboard macro defined.

`euporie.commands.buffer.select_all()` → `None`
Select all text.

`euporie.commands.buffer.show_contextual_help(event: KeyPressEvent)` → `None`
Displays contextual help.

`euporie.commands.buffer.start_macro()` → `None`
Start recording a macro.

`euporie.commands.buffer.start_selection(event: KeyPressEvent)` → `None`
Start a new selection.

`euporie.commands.buffer.toggle_case()` → `None`
Toggle the case of the current word or selection.

`euporie.commands.buffer.toggle_comment()` → `None`
Comments or uncomments the current or selected lines.

`euporie.commands.buffer.toggle_overwrite_mode()` → `None`
Toggle overwrite when using micro editing mode.

`euporie.commands.buffer.type_key(event: KeyPressEvent)` → `None`
Enter a key.

`euporie.commands.buffer.undo()` → `None`
Undo the last edit.

`euporie.commands.buffer.unindent_lines(event: KeyPressEvent)` → `None`
Unindent the current or selected lines.

`euporie.commands.buffer.unshift_move(event: KeyPressEvent)` → `None`
Used for the shift selection mode.

When called with a shift + movement key press event, moves the cursor as if shift is not pressed.

Parameters `event` – The key press event to process

`euporie.commands.buffer.wrap_selection_cmd(left: str, right: str) → None`
 Adds strings to either end of the current selection.

euporie.commands.cell

Defines commands relating to cells.

Functions

<code>edit_in_external_editor()</code>	Edit cell in \$EDITOR.
<code>enter_cell_edit_mode()</code>	Enter cell edit mode.
<code>exit_edit_mode()</code>	Exit cell edit mode.
<code>split_cell()</code>	Split the current cell at the cursor position.

euporie.commands.cell.edit_in_external_editor

`async euporie.commands.cell.edit_in_external_editor() → None`
 Edit cell in \$EDITOR.

euporie.commands.cell.enter_cell_edit_mode

`euporie.commands.cell.enter_cell_edit_mode() → None`
 Enter cell edit mode.

euporie.commands.cell.exit_edit_mode

`euporie.commands.cell.exit_edit_mode() → None`
 Exit cell edit mode.

euporie.commands.cell.split_cell

`euporie.commands.cell.split_cell() → None`
 Split the current cell at the cursor position.

`async euporie.commands.cell.edit_in_external_editor() → None`
 Edit cell in \$EDITOR.

`euporie.commands.cell.enter_cell_edit_mode() → None`
 Enter cell edit mode.

`euporie.commands.cell.exit_edit_mode() → None`
 Exit cell edit mode.

`euporie.commands.cell.split_cell() → None`
 Split the current cell at the cursor position.

euporie.commands.cell_output

Defines commands relating to cell outputs.

Functions

<code>go_to_end_of_cell_output()</code>	Scroll the output down one page.
<code>go_to_start_of_cell_output()</code>	Scroll the output to the top.
<code>page_down_cell_output()</code>	Scroll the output down one page.
<code>page_up_cell_output()</code>	Scroll the output up one page.
<code>scroll_down_cell_output()</code>	Scroll the output down one line.
<code>scroll_up_cell_output()</code>	Scroll the output up one line.

euporie.commands.cell_output.go_to_end_of_cell_output

`euporie.commands.cell_output.go_to_end_of_cell_output()` → [None](#)
Scroll the output down one page.

euporie.commands.cell_output.go_to_start_of_cell_output

`euporie.commands.cell_output.go_to_start_of_cell_output()` → [None](#)
Scroll the output to the top.

euporie.commands.cell_output.page_down_cell_output

`euporie.commands.cell_output.page_down_cell_output()` → [None](#)
Scroll the output down one page.

euporie.commands.cell_output.page_up_cell_output

`euporie.commands.cell_output.page_up_cell_output()` → [None](#)
Scroll the output up one page.

euporie.commands.cell_output.scroll_down_cell_output

`euporie.commands.cell_output.scroll_down_cell_output()` → [None](#)
Scroll the output down one line.

euporie.commands.cell_output.scroll_up_cell_output

`euporie.commands.cell_output.scroll_up_cell_output()` → None

Scroll the output up one line.

`euporie.commands.cell_output.go_to_end_of_cell_output()` → None

Scroll the output down one page.

`euporie.commands.cell_output.go_to_start_of_cell_output()` → None

Scroll the output to the top.

`euporie.commands.cell_output.page_down_cell_output()` → None

Scroll the output down one page.

`euporie.commands.cell_output.page_up_cell_output()` → None

Scroll the output up one page.

`euporie.commands.cell_output.scroll_down_cell_output()` → None

Scroll the output down one line.

`euporie.commands.cell_output.scroll_up_cell_output()` → None

Scroll the output up one line.

euporie.commands.completions

Defines commands relating to completions.

Functions

<code>accept_completion()</code>	Accept a selected completion.
<code>cancel_completion()</code>	Cancel a completion.

euporie.commands.completions.accept_completion

`euporie.commands.completions.accept_completion()` → None

Accept a selected completion.

euporie.commands.completions.cancel_completion

`euporie.commands.completions.cancel_completion()` → None

Cancel a completion.

`euporie.commands.completions.accept_completion()` → None

Accept a selected completion.

`euporie.commands.completions.cancel_completion()` → None

Cancel a completion.

euporie.commands.config

Define commands at the application level.

Functions

<code>autocomplete()</code>	Toggle whether completions should be shown automatically.
<code>autoformat()</code>	Toggle whether code cells are formatted before they are run.
<code>autoinspect()</code>	Toggle whether to automatically show contextual help when navigating code cells.
<code>autosuggest()</code>	Toggle whether to suggest line completions from the kernel's history.
<code>format_black()</code>	Toggle whether code cells are formatted using black.
<code>format_isort()</code>	Toggle whether code cells are formatted using isort.
<code>format_ssort()</code>	Toggle whether code cells are formatted using ssort.
<code>run_after_external_edit()</code>	Toggle whether cells should run automatically after editing externally.
<code>set_edit_mode(edit_mode)</code>	Set the editing mode key-binding style.
<code>show_cell_borders()</code>	Toggle the visibility of the borders of unselected cells.
<code>show_line_numbers()</code>	Toggle the visibility of line numbers.
<code>show_status_bar()</code>	Toggle the visibility of the status bar.
<code>switch_background_pattern()</code>	Switch between different background patterns.
<code>tmux_terminal_graphics()</code>	Toggle the use of terminal graphics inside tmux.
<code>update_color_scheme(choice)</code>	Updates the application's style.
<code>update_syntax_theme(choice)</code>	Updates the application's syntax highlighting theme.
<code>use_full_width()</code>	Toggle whether cells should extend across the full width of the screen.

euporie.commands.config.autocomplete

`euporie.commands.config.autocomplete()` → `None`
Toggle whether completions should be shown automatically.

euporie.commands.config.autoformat

`euporie.commands.config.autoformat()` → `None`
Toggle whether code cells are formatted before they are run.

euporie.commands.config.autoinspect

`euporie.commands.config.autoinspect()` → `None`

Toggle whether to automatically show contextual help when navigating code cells.

euporie.commands.config.autosuggest

`euporie.commands.config.autosuggest()` → `None`

Toggle whether to suggest line completions from the kernel's history.

euporie.commands.config.format_black

`euporie.commands.config.format_black()` → `None`

Toggle whether code cells are formatted using black.

euporie.commands.config.format_isort

`euporie.commands.config.format_isort()` → `None`

Toggle whether code cells are formatted using isort.

euporie.commands.config.format_ssort

`euporie.commands.config.format_ssort()` → `None`

Toggle whether code cells are formatted using ssort.

euporie.commands.config.run_after_external_edit

`euporie.commands.config.run_after_external_edit()` → `None`

Toggle whether cells should run automatically after editing externally.

euporie.commands.config.set_edit_mode

`euporie.commands.config.set_edit_mode(edit_mode: EditingMode)` → `None`

Set the editing mode key-binding style.

euporie.commands.config.show_cell_borders

`euporie.commands.config.show_cell_borders()` → `None`

Toggle the visibility of the borders of unselected cells.

euporie.commands.config.show_line_numbers

`euporie.commands.config.show_line_numbers()` → `None`

Toggle the visibility of line numbers.

euporie.commands.config.show_status_bar

`euporie.commands.config.show_status_bar()` → `None`

Toggle the visibility of the status bar.

euporie.commands.config.switch_background_pattern

`euporie.commands.config.switch_background_pattern()` → `None`

Switch between different background patterns.

euporie.commands.config.tmux_terminal_graphics

`euporie.commands.config.tmux_terminal_graphics()` → `None`

Toggle the use of terminal graphics inside tmux.

euporie.commands.config.update_color_scheme

`euporie.commands.config.update_color_scheme(choice: str)` → `None`

Updates the application's style.

euporie.commands.config.update_syntax_theme

`euporie.commands.config.update_syntax_theme(choice: str)` → `None`

Updates the application's syntax highlighting theme.

euporie.commands.config.use_full_width

`euporie.commands.config.use_full_width()` → `None`

Toggle whether cells should extend across the full width of the screen.

`euporie.commands.config.autocomplete()` → `None`

Toggle whether completions should be shown automatically.

`euporie.commands.config.autoformat()` → `None`

Toggle whether code cells are formatted before they are run.

`euporie.commands.config.autoinspect()` → `None`

Toggle whether to automatically show contextual help when navigating code cells.

`euporie.commands.config.autosuggest()` → `None`

Toggle whether to suggest line completions from the kernel's history.

`euporie.commands.config.format_black()` → `None`

Toggle whether code cells are formatted using black.

`euporie.commands.config.format_isort()` → `None`
Toggle whether code cells are formatted using isort.

`euporie.commands.config.format_ssort()` → `None`
Toggle whether code cells are formatted using ssort.

`euporie.commands.config.run_after_external_edit()` → `None`
Toggle whether cells should run automatically after editing externally.

`euporie.commands.config.set_edit_mode(edit_mode: EditingMode)` → `None`
Set the editing mode key-binding style.

`euporie.commands.config.show_cell_borders()` → `None`
Toggle the visibility of the borders of unselected cells.

`euporie.commands.config.show_line_numbers()` → `None`
Toggle the visibility of line numbers.

`euporie.commands.config.show_status_bar()` → `None`
Toggle the visibility of the status bar.

`euporie.commands.config.switch_background_pattern()` → `None`
Switch between different background patterns.

`euporie.commands.config.tmux_terminal_graphics()` → `None`
Toggle the use of terminal graphics inside tmux.

`euporie.commands.config.update_color_scheme(choice: str)` → `None`
Updates the application's style.

`euporie.commands.config.update_syntax_theme(choice: str)` → `None`
Updates the application's syntax highlighting theme.

`euporie.commands.config.use_full_width()` → `None`
Toggle whether cells should extend across the full width of the screen.

euporie.commands.format

Define commands at the application level.

Functions

<code>format_command_attrs([groups, attrs, pad, ...])</code>	Format command attributes by group.
--	-------------------------------------

euporie.commands.format.format_command_attrs

`euporie.commands.format.format_command_attrs(groups: Optional[list[str]] = None, attrs: Optional[list[str]] = None, pad: bool = True, escape: bool = False)` → `dict[str, list[dict[str, Union[str, list[str]]]]]`

Format command attributes by group.

Parameters

- **groups** – List of command groups to include. If `None`, includes all groups
- **attrs** – List of command attributes to include. If `None`, includes all attributes

- **pad** – Whether to pad attribute strings to equal lengths
- **escape** – Whether to escape back-ticks in strings

Returns A dictionary mapping group names to lists of command details

`euporie.commands.format.format_command_attrs`(*groups: Optional[list[str]] = None, attrs: Optional[list[str]] = None, pad: bool = True, escape: bool = False*) → dict[str, list[dict[str, Union[str, list[str]]]]]

Format command attributes by group.

Parameters

- **groups** – List of command groups to include. If None, includes all groups
- **attrs** – List of command attributes to include. If None, includes all attributes
- **pad** – Whether to pad attribute strings to equal lengths
- **escape** – Whether to escape back-ticks in strings

Returns A dictionary mapping group names to lists of command details

euporie.commands.notebook

Defines commands relating to notebooks.

Functions

<code>add_cell_above()</code>	Add a new cell above the current.
<code>add_cell_below()</code>	Add a new cell below the current.
<code>cells_to_code()</code>	Change selected cells to code cells.
<code>cells_to_markdown()</code>	Change selected cells to markdown cells.
<code>cells_to_raw()</code>	Change selected cells to raw cells.
<code>change_kernel()</code>	Change the notebook's kernel.
<code>copy_cells()</code>	Copy the current cells.
<code>cut_cells()</code>	Cut the current cells.
<code>delete_cells()</code>	Delete the current cells.
<code>extend_cell_selection_down()</code>	Go up one cell.
<code>extend_cell_selection_up()</code>	Go up one cell.
<code>interrupt_kernel()</code>	Interrupt the notebook's kernel.
<code>merge_cells()</code>	Merge the selected cells.
<code>move_cells_down()</code>	Move selected cells down.
<code>move_cells_up()</code>	Move selected cells up.
<code>next_child()</code>	Select the next cell.
<code>paste_cells()</code>	Paste the previously copied cells.
<code>reformat_cells()</code>	Format the selected code cells.
<code>reformat_notebook()</code>	Automatically reformat all code cells in the notebook.
<code>restart_kernel()</code>	Restart the notebook's kernel.
<code>run_all_cells()</code>	Run or render all the cells in the current notebook.
<code>run_cell_and_insert_below()</code>	Run or render the current cells and insert a new cell below.
<code>run_selected_cells()</code>	Run or render the current cells.

continues on next page

Table 21 – continued from previous page

<code>run_selected_cells_and_select_next_cell()</code>	Run or render the current cells and select the next cell.
<code>save_notebook()</code>	Save the current notebook.
<code>scroll_down()</code>	Scroll the page down a line.
<code>scroll_down_5_lines()</code>	Scroll the page down 5 lines.
<code>scroll_up()</code>	Scroll the page up a line.
<code>scroll_up_5_lines()</code>	Scroll the page up 5 lines.
<code>select_5th_next_cell()</code>	Go down 5 cells.
<code>select_5th_previous_cell()</code>	Go up 5 cells.
<code>select_all_cells()</code>	Select all cells in the notebook.
<code>select_first_cell()</code>	Select the first cell in the notebook.
<code>select_last_cell()</code>	Select the last cell in the notebook.
<code>select_previous_cell()</code>	Go up one cell.

euporie.commands.notebook.add_cell_above

`euporie.commands.notebook.add_cell_above()` → `None`
 Add a new cell above the current.

euporie.commands.notebook.add_cell_below

`euporie.commands.notebook.add_cell_below()` → `None`
 Add a new cell below the current.

euporie.commands.notebook.cells_to_code

`euporie.commands.notebook.cells_to_code()` → `None`
 Change selected cells to code cells.

euporie.commands.notebook.cells_to_markdown

`euporie.commands.notebook.cells_to_markdown()` → `None`
 Change selected cells to markdown cells.

euporie.commands.notebook.cells_to_raw

`euporie.commands.notebook.cells_to_raw()` → `None`
 Change selected cells to raw cells.

euporie.commands.notebook.change_kernel

`euporie.commands.notebook.change_kernel()` → `None`
Change the notebook's kernel.

euporie.commands.notebook.copy_cells

`euporie.commands.notebook.copy_cells()` → `None`
Copy the current cells.

euporie.commands.notebook.cut_cells

`euporie.commands.notebook.cut_cells()` → `None`
Cut the current cells.

euporie.commands.notebook.delete_cells

`euporie.commands.notebook.delete_cells()` → `None`
Delete the current cells.

euporie.commands.notebook.extend_cell_selection_down

`euporie.commands.notebook.extend_cell_selection_down()` → `None`
Go up one cell.

euporie.commands.notebook.extend_cell_selection_up

`euporie.commands.notebook.extend_cell_selection_up()` → `None`
Go up one cell.

euporie.commands.notebook.interrupt_kernel

`euporie.commands.notebook.interrupt_kernel()` → `None`
Interrupt the notebook's kernel.

euporie.commands.notebook.merge_cells

`euporie.commands.notebook.merge_cells()` → `None`
Merge the selected cells.

euporie.commands.notebook.move_cells_down

`euporie.commands.notebook.move_cells_down()` → `None`
Move selected cells down.

euporie.commands.notebook.move_cells_up

`euporie.commands.notebook.move_cells_up()` → `None`
Move selected cells up.

euporie.commands.notebook.next_child

`euporie.commands.notebook.next_child()` → `None`
Select the next cell.

euporie.commands.notebook.paste_cells

`euporie.commands.notebook.paste_cells()` → `None`
Paste the previously copied cells.

euporie.commands.notebook.reformat_cells

`euporie.commands.notebook.reformat_cells()` → `None`
Format the selected code cells.

euporie.commands.notebook.reformat_notebook

`euporie.commands.notebook.reformat_notebook()` → `None`
Automatically reformat all code cells in the notebook.

euporie.commands.notebook.restart_kernel

`euporie.commands.notebook.restart_kernel()` → `None`
Restart the notebook's kernel.

euporie.commands.notebook.run_all_cells

`euporie.commands.notebook.run_all_cells()` → `None`
Run or render all the cells in the current notebook.

euporie.commands.notebook.run_cell_and_insert_below

`euporie.commands.notebook.run_cell_and_insert_below()` → `None`

Run or render the current cells and insert a new cell below.

euporie.commands.notebook.run_selected_cells

`euporie.commands.notebook.run_selected_cells()` → `None`

Run or render the current cells.

euporie.commands.notebook.run_selected_cells_and_select_next_cell

`euporie.commands.notebook.run_selected_cells_and_select_next_cell()` → `None`

Run or render the current cells and select the next cell.

euporie.commands.notebook.save_notebook

`euporie.commands.notebook.save_notebook()` → `None`

Save the current notebook.

euporie.commands.notebook.scroll_down

`euporie.commands.notebook.scroll_down()` → `None`

Scroll the page down a line.

euporie.commands.notebook.scroll_down_5_lines

`euporie.commands.notebook.scroll_down_5_lines()` → `None`

Scroll the page down 5 lines.

euporie.commands.notebook.scroll_up

`euporie.commands.notebook.scroll_up()` → `None`

Scroll the page up a line.

euporie.commands.notebook.scroll_up_5_lines

`euporie.commands.notebook.scroll_up_5_lines()` → `None`

Scroll the page up 5 lines.

euporie.commands.notebook.select_5th_next_cell

`euporie.commands.notebook.select_5th_next_cell()` → `None`
Go down 5 cells.

euporie.commands.notebook.select_5th_previous_cell

`euporie.commands.notebook.select_5th_previous_cell()` → `None`
Go up 5 cells.

euporie.commands.notebook.select_all_cells

`euporie.commands.notebook.select_all_cells()` → `None`
Select all cells in the notebook.

euporie.commands.notebook.select_first_cell

`euporie.commands.notebook.select_first_cell()` → `None`
Select the first cell in the notebook.

euporie.commands.notebook.select_last_cell

`euporie.commands.notebook.select_last_cell()` → `None`
Select the last cell in the notebook.

euporie.commands.notebook.select_previous_cell

`euporie.commands.notebook.select_previous_cell()` → `None`
Go up one cell.

`euporie.commands.notebook.add_cell_above()` → `None`
Add a new cell above the current.

`euporie.commands.notebook.add_cell_below()` → `None`
Add a new cell below the current.

`euporie.commands.notebook.cells_to_code()` → `None`
Change selected cells to code cells.

`euporie.commands.notebook.cells_to_markdown()` → `None`
Change selected cells to markdown cells.

`euporie.commands.notebook.cells_to_raw()` → `None`
Change selected cells to raw cells.

`euporie.commands.notebook.change_kernel()` → `None`
Change the notebook's kernel.

`euporie.commands.notebook.copy_cells()` → `None`
Copy the current cells.

`euporie.commands.notebook.cut_cells()` → `None`
Cut the current cells.

`euporie.commands.notebook.delete_cells()` → `None`
Delete the current cells.

`euporie.commands.notebook.extend_cell_selection_down()` → `None`
Go up one cell.

`euporie.commands.notebook.extend_cell_selection_up()` → `None`
Go up one cell.

`euporie.commands.notebook.interrupt_kernel()` → `None`
Interrupt the notebook's kernel.

`euporie.commands.notebook.merge_cells()` → `None`
Merge the selected cells.

`euporie.commands.notebook.move_cells_down()` → `None`
Move selected cells down.

`euporie.commands.notebook.move_cells_up()` → `None`
Move selected cells up.

`euporie.commands.notebook.next_child()` → `None`
Select the next cell.

`euporie.commands.notebook.paste_cells()` → `None`
Paste the previously copied cells.

`euporie.commands.notebook.reformat_cells()` → `None`
Format the selected code cells.

`euporie.commands.notebook.reformat_notebook()` → `None`
Automatically reformat all code cells in the notebook.

`euporie.commands.notebook.restart_kernel()` → `None`
Restart the notebook's kernel.

`euporie.commands.notebook.run_all_cells()` → `None`
Run or render all the cells in the current notebook.

`euporie.commands.notebook.run_cell_and_insert_below()` → `None`
Run or render the current cells and insert a new cell below.

`euporie.commands.notebook.run_selected_cells()` → `None`
Run or render the current cells.

`euporie.commands.notebook.run_selected_cells_and_select_next_cell()` → `None`
Run or render the current cells and select the next cell.

`euporie.commands.notebook.save_notebook()` → `None`
Save the current notebook.

`euporie.commands.notebook.scroll_down()` → `None`
Scroll the page down a line.

`euporie.commands.notebook.scroll_down_5_lines()` → `None`
Scroll the page down 5 lines.

`euporie.commands.notebook.scroll_up()` → `None`
Scroll the page up a line.

`euporie.commands.notebook.scroll_up_5_lines()` → `None`
Scroll the page up 5 lines.

`euporie.commands.notebook.select_5th_next_cell()` → `None`
Go down 5 cells.

`euporie.commands.notebook.select_5th_previous_cell()` → `None`
Go up 5 cells.

`euporie.commands.notebook.select_all_cells()` → `None`
Select all cells in the notebook.

`euporie.commands.notebook.select_first_cell()` → `None`
Select the first cell in the notebook.

`euporie.commands.notebook.select_last_cell()` → `None`
Select the last cell in the notebook.

`euporie.commands.notebook.select_previous_cell()` → `None`
Go up one cell.

euporie.commands.pager

Defines commands for the pager.

Functions

<code>close_pager()</code>	Close the pager.
----------------------------	------------------

euporie.commands.pager.close_pager

`euporie.commands.pager.close_pager()` → `None`
Close the pager.

`euporie.commands.pager.close_pager()` → `None`
Close the pager.

euporie.commands.registry

Defines functions to manage the command registry.

Functions

<code>add(**kwargs)</code>	Adds a command to the centralized command system.
<code>get(name)</code>	Get a command from the centralized command system by name.

euporie.commands.registry.add

`euporie.commands.registry.add(**kwargs: Any) → Callable`
Adds a command to the centralized command system.

euporie.commands.registry.get

`euporie.commands.registry.get(name: str) → euporie.commands.base.Command`
Get a command from the centralized command system by name.

`euporie.commands.registry.add(**kwargs: Any) → Callable`
Adds a command to the centralized command system.

`euporie.commands.registry.get(name: str) → euporie.commands.base.Command`
Get a command from the centralized command system by name.

euporie.commands.search

Defines commands related to searching.

euporie.commands.suggestions

Defines command relating to suggestions.

Functions

<code>accept_suggestion(event)</code>	Accept suggestion.
<code>fill_suggestion(event)</code>	Fill partial suggestion.

euporie.commands.suggestions.accept_suggestion

`euporie.commands.suggestions.accept_suggestion(event: KeyPressEvent) → None`
Accept suggestion.

euporie.commands.suggestions.fill_suggestion

`euporie.commands.suggestions.fill_suggestion(event: KeyPressEvent) → None`
Fill partial suggestion.

`euporie.commands.suggestions.accept_suggestion(event: KeyPressEvent) → None`
Accept suggestion.

`euporie.commands.suggestions.fill_suggestion(event: KeyPressEvent) → None`
Fill partial suggestion.

euporie.commands.tui

Define commands at the application level.

Functions

<i>about()</i>	Show the about dialog.
<i>close_file()</i>	Close the current file.
<i>focus_next()</i>	Focus the next control.
<i>focus_previous()</i>	Focus the previous control.
<i>keyboard_shortcuts()</i>	Show the currently bound keyboard shortcuts.
<i>new_notebook()</i>	Create a new file.
<i>next_tab()</i>	Switch to the next tab.
<i>open_file()</i>	Open a file.
<i>previous_tab()</i>	Switch to the previous tab.
<i>quit()</i>	Quit euporie.
<i>show_command_palette()</i>	Shows the command palette.
<i>view_documentation()</i>	Open the documentation in the browser.
<i>view_logs()</i>	Open the logs in a new tab.

euporie.commands.tui.about

`euporie.commands.tui.about()` → `None`
Show the about dialog.

euporie.commands.tui.close_file

`euporie.commands.tui.close_file()` → `None`
Close the current file.

euporie.commands.tui.focus_next

`euporie.commands.tui.focus_next()` → `None`
Focus the next control.

euporie.commands.tui.focus_previous

`euporie.commands.tui.focus_previous()` → `None`
Focus the previous control.

euporie.commands.tui.keyboard_shortcuts

`euporie.commands.tui.keyboard_shortcuts()` → `None`
Show the currently bound keyboard shortcuts.

euporie.commands.tui.new_notebook

`euporie.commands.tui.new_notebook()` → `None`
Create a new file.

euporie.commands.tui.next_tab

`euporie.commands.tui.next_tab()` → `None`
Switch to the next tab.

euporie.commands.tui.open_file

`euporie.commands.tui.open_file()` → `None`
Open a file.

euporie.commands.tui.previous_tab

`euporie.commands.tui.previous_tab()` → `None`
Switch to the previous tab.

euporie.commands.tui.quit

`euporie.commands.tui.quit()` → `None`
Quit euporie.

euporie.commands.tui.show_command_palette

`euporie.commands.tui.show_command_palette()` → `None`
Shows the command palette.

euporie.commands.tui.view_documentation

`euporie.commands.tui.view_documentation()` → `None`
Open the documentation in the browser.

euporie.commands.tui.view_logs**euporie.commands.tui.view_logs()** → *None*

Open the logs in a new tab.

euporie.commands.tui.about() → *None*

Show the about dialog.

euporie.commands.tui.close_file() → *None*

Close the current file.

euporie.commands.tui.focus_next() → *None*

Focus the next control.

euporie.commands.tui.focus_previous() → *None*

Focus the previous control.

euporie.commands.tui.keyboard_shortcuts() → *None*

Show the currently bound keyboard shortcuts.

euporie.commands.tui.new_notebook() → *None*

Create a new file.

euporie.commands.tui.next_tab() → *None*

Switch to the next tab.

euporie.commands.tui.open_file() → *None*

Open a file.

euporie.commands.tui.previous_tab() → *None*

Switch to the previous tab.

euporie.commands.tui.quit() → *None*

Quit euporie.

euporie.commands.tui.show_command_palette() → *None*

Shows the command palette.

euporie.commands.tui.view_documentation() → *None*

Open the documentation in the browser.

euporie.commands.tui.view_logs() → *None*

Open the logs in a new tab.

class euporie.commands.**Command**(*handler: Callable[...]*, *Optional[Awaitable[Any]]*, *, *filter: FilterOrBool = True*, *hidden: FilterOrBool = False*, *name: Optional[str] = None*, *title: Optional[str] = None*, *menu_title: Optional[str] = None*, *description: Optional[str] = None*, *group: Optional[str] = None*, *toggled: Optional[Filter] = None*, *keys: Optional[AnyKeys] = None*, *eager: FilterOrBool = False*, *is_global: FilterOrBool = False*, *save_before: Callable[[KeyPressEvent], bool] = <function Command.<lambda>>*, *record_in_macro: FilterOrBool = True*)

Bases: **object**

Wraps a function so it can be used as a key-binding or a menu item.

add_keys(*keys: Optional[AnyKeys]*) → *Command*

Adds keyboard shortcuts to the current command.

bind(*key_bindings: KeyBindingsBase*, *keys: Optional[AnyKeys] = None*) → *None*

Add the current commands to a set of key bindings.

Parameters

- **key_bindings** – The set of key bindings to bind to
- **keys** – Additional keys to bind to the command

property key_bindings: Sequence[Binding]

Returns a list of key-bindings given to the current command.

property key_handler: KeyHandlerCallable

Returns a key handler for the command.

property menu: euporie.menu.item.MenuItem

Returns a menu item for the command.

property menu_handler: Callable[[], None]

Returns a menu handler for the command.

run() → None

Runs the command's handler.

euporie.commands.add(**kwargs: Any) → Callable

Adds a command to the centralized command system.

euporie.commands.get(name: str) → euporie.commands.base.Command

Get a command from the centralized command system by name.

5.8.5 euporie.completion

Contains the main class for a notebook file.

Classes

KernelCompleter(kernel)

A prompt_toolkit completer which provides completions from a Jupyter kernel.

euporie.completion.KernelCompleter

class euporie.completion.KernelCompleter(kernel: euporie.kernel.NotebookKernel)

A prompt_toolkit completer which provides completions from a Jupyter kernel.

class euporie.completion.KernelCompleter(kernel: euporie.kernel.NotebookKernel)

Bases: prompt_toolkit.completion.base.Completer

A prompt_toolkit completer which provides completions from a Jupyter kernel.

get_completions(document: prompt_toolkit.document.Document, complete_event:

prompt_toolkit.completion.base.CompleteEvent) →

Iterable[prompt_toolkit.completion.base.Completion]

Does nothing as completions are retrieved asynchronously.

async get_completions_async(document: prompt_toolkit.document.Document, complete_event:

prompt_toolkit.completion.base.CompleteEvent) →

AsyncGenerator[prompt_toolkit.completion.base.Completion, None]

Retrieves completions from a NotebookKernel.

5.8.6 euporie.config

Defines a configuration class for euporie.

Classes

<code>BooleanOptionalAction(option_strings, *args, ...)</code>	Action for boolean flags.
<code>Config()</code>	A configuration object with configuration values available as attributes.
<code>JSONEncoderPlus(*[, skipkeys, ensure_ascii, ...])</code>	JSON encode class which encodes paths as strings.

euporie.config.BooleanOptionalAction

class euporie.config.BooleanOptionalAction(option_strings: list[str], *args: Any, **kwargs: Any)

Action for boolean flags.

Included because *argparse.BooleanOptionalAction* is not present in *python<=3.9*.

euporie.config.Config

class euporie.config.Config

A configuration object with configuration values available as attributes.

Default configuration variables are loaded from the defaults defined in the schema, then overwritten with values defined in a configuration file.

euporie.config.JSONEncoderPlus

class euporie.config.JSONEncoderPlus(*, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None)

JSON encode class which encodes paths as strings.

class euporie.config.BooleanOptionalAction(option_strings: list[str], *args: Any, **kwargs: Any)

Bases: `argparse.Action`

Action for boolean flags.

Included because *argparse.BooleanOptionalAction* is not present in *python<=3.9*.

format_usage() → str

Formats the action string.

Returns The formatted string.

class euporie.config.Config

Bases: `object`

A configuration object with configuration values available as attributes.

Default configuration variables are loaded from the defaults defined in the schema, then overwritten with values defined in a configuration file.

choices(name: str) → list

Returns a list of valid choices for a configuration item.

Parameters *name* – The name of the attribute to query.

Returns A list of valid choices

`conf_file_name = 'config.json'`

```
defaults = {'autocomplete': False, 'autoformat': False, 'autoinspect': False,
'autosuggest': True, 'background_character': '.', 'background_color': '',
'background_pattern': 2, 'color_scheme': 'default', 'debug': False, 'dump':
False, 'dump_file': None, 'edit_mode': 'micro', 'expand': False, 'files': [],
'format_black': True, 'format_isort': True, 'format_ssort': True, 'line_numbers':
True, 'log_file': '', 'max_notebook_width': 120, 'page': False, 'run': False,
'run_after_external_edit': False, 'show_cell_borders': False, 'show_status_bar':
True, 'syntax_theme': 'default', 'tab_size': 4, 'terminal_polling_interval': 0,
'tmux_graphics': False, 'version': None}
```

`get(name: str) → Any`

Access a configuration variable, falling back to the default value if unset.

Parameters *name* – The name of the attribute to access.

Returns The configuration variable value.

`load() → euporie.config.Config`

Loads the command line, environment, and user configuration.

`load_args() → None`

Attempts to load configuration settings from commandline flags.

`load_env() → None`

Attempt to load configuration settings from environment variables.

`load_parser() → argparse.ArgumentParser`

Constructs an `argparse.ArgumentParser`.

`load_user() → None`

Attempt to load JSON configuration file.

`toggle(name: str) → None`

Switches attributes between permitted configuration states.

For boolean values, they are toggled between True and False. Integer values are incremented and reset within the permitted range.

Parameters *name* – The name of the attribute to toggle.

`valid_user: bool`

```
class euporie.config.JSONEncoderPlus(*, skipkeys=False, ensure_ascii=True, check_circular=True,
allow_nan=True, sort_keys=False, indent=None, separators=None,
default=None)
```

Bases: `json.encoder.JSONEncoder`

JSON encode class which encodes paths as strings.

`default(o: Any) → Union[bool, int, float, str, None]`

Encode an object to JSON.

Parameters *o* – The object to encode

Returns The encoded object

`encode(o)`

Return a JSON string representation of a Python data structure.

```
>>> from json.encoder import JSONEncoder
>>> JSONEncoder().encode({"foo": ["bar", "baz"]})
'{"foo": ["bar", "baz"]}'
```

```
item_separator = ', '
```

```
iterencode(o, _one_shot=False)
```

Encode the given object and yield each string representation as available.

For example:

```
for chunk in JSONEncoder().iterencode(bigobject):
    mysocket.write(chunk)
```

```
key_separator = ': '
```

5.8.7 euporie.containers

Contains the *ScrollingContainer* class, which renders children on the fly.

Classes

<i>PrintingContainer</i> (children[, width])	A container which displays all it's children in a vertical list.
--	--

euporie.containers.PrintingContainer

```
class euporie.containers.PrintingContainer(children: Union[Callable, Sequence[AnyContainer]], width: AnyDimension = None)
```

A container which displays all it's children in a vertical list.

```
class euporie.containers.PrintingContainer(children: Union[Callable, Sequence[AnyContainer]], width: AnyDimension = None)
```

Bases: `prompt_toolkit.layout.containers.Container`

A container which displays all it's children in a vertical list.

property children: `Sequence[AnyContainer]`

Returns the container's children.

get_children() → `list[prompt_toolkit.layout.containers.Container]`

Returns a list of all child containers.

get_key_bindings() → `Optional[prompt_toolkit.key_binding.key_bindings.KeyBindingsBase]`

Returns a KeyBindings object. These bindings become active when any user control in this container has the focus, except if any containers between this container and the focused user control is modal.

is_modal() → `bool`

When this container is modal, key bindings from parent containers are not taken into account if a user control in this container is focused.

preferred_height(width: `int`, max_available_height: `int`) → `prompt_toolkit.layout.dimension.Dimension`

Returns the preferred height, equal to the sum of the child heights.

preferred_width(*max_available_width: int*) → `prompt_toolkit.layout.dimension.Dimension`

Calculates and returns the desired width for this container.

reset() → `None`

Reset the state of this container and all the children.

Does nothing as this container is used for dumping output.

write_to_screen(*screen: Screen, mouse_handlers: MouseHandlers, write_position: WritePosition, parent_style: str, erase_bg: bool, z_index: Optional[int]*) → `None`

Render the container to a *Screen* instance.

All children are rendered vertically in sequence.

Parameters

- **screen** – The `Screen` class to which the output has to be written.
- **mouse_handlers** – `prompt_toolkit.layout.mouse_handlers.MouseHandlers`.
- **write_position** – A `prompt_toolkit.layout.screen.WritePosition` object defining where this container should be drawn.
- **erase_bg** – If true, the background will be erased prior to drawing.
- **parent_style** – Style string to pass to the Window object. This will be applied to all content of the windows. `VSplit` and `prompt_toolkit.layout.containers.HSplit` can use it to pass their style down to the windows that they contain.
- **z_index** – Used for propagating `z_index` from parent to child.

5.8.8 euporie.convert

Sub-module concerned with the conversion of data formats.

Modules

<code>euporie.convert.base</code>	Contains main format conversion function.
<code>euporie.convert.formats</code>	Contains various data format conversion functions.
<code>euporie.convert.util</code>	Utility functions for format converters.

euporie.convert.base

Contains main format conversion function.

Functions

<code>convert</code> (data, from_, to[, cols, rows, fg, bg])	Convert between formats.
<code>find_route</code> (from_, to)	Finds the shortest conversion path between two formats.
<code>register</code> (from_, to[, filter_, weight])	Adds a converter to the centralized format conversion system.

euporie.convert.base.convert

`euporie.convert.base.convert`(*data*: *str*, *from_*: *str*, *to*: *str*, *cols*: *Optional[int]* = *None*, *rows*: *Optional[int]* = *None*, *fg*: *Optional[str]* = *None*, *bg*: *Optional[str]* = *None*) → *Any*

Convert between formats.

euporie.convert.base.find_route

`euporie.convert.base.find_route`(*from_*: *str*, *to*: *str*) → *Optional[list]*

Finds the shortest conversion path between two formats.

euporie.convert.base.register

`euporie.convert.base.register`(*from_*: *Union[Iterable[str], str]*, *to*: *str*, *filter_*: *FilterOrBool* = *True*, *weight*: *int* = *1*) → *Callable*

Adds a convertor to the centralized format conversion system.

Classes

<i>Convertor</i> (<i>func</i> [, <i>weight</i>])	Holds a conversion function and its weight.
--	---

euporie.convert.base.Convertor

class `euporie.convert.base.Convertor`(*func*: *Callable*, *weight*: *int* = *1*)

Holds a conversion function and its weight.

class `euporie.convert.base.Convertor`(*func*: *Callable*, *weight*: *int* = *1*)

Bases: `NamedTuple`

Holds a conversion function and its weight.

count(*value*, /)

Return number of occurrences of value.

func: `Callable`

Alias for field number 0

index(*value*, *start*=0, *stop*=9223372036854775807, /)

Return first index of value.

Raises `ValueError` if the value is not present.

weight: `int`

Alias for field number 1

`euporie.convert.base.convert`(*data*: *str*, *from_*: *str*, *to*: *str*, *cols*: *Optional[int]* = *None*, *rows*: *Optional[int]* = *None*, *fg*: *Optional[str]* = *None*, *bg*: *Optional[str]* = *None*) → *Any*

Convert between formats.

`euporie.convert.base.find_route`(*from_*: *str*, *to*: *str*) → *Optional[list]*

Finds the shortest conversion path between two formats.

`euporie.convert.base.register`(*from_*: Union[Iterable[str], str], *to*: str, *filter_*: FilterOrBool = True, *weight*: int = 1) → Callable

Adds a convertor to the centralized format conversion system.

euporie.convert.formats

Contains various data format conversion functions.

They are grouped into sub-modules based on output format.

Modules

<code>euporie.convert.formats.ansi</code>	Contains functions which convert data to formatted ansi text.
<code>euporie.convert.formats.base64</code>	Contains functions which convert data to base64 format.
<code>euporie.convert.formats.common</code>	Contains functions which can be used to convert data to multiple formats.
<code>euporie.convert.formats.jpeg</code>	Contains functions which convert data to jpeg format.
<code>euporie.convert.formats.markdown</code>	Contains functions which convert data to markdown format.
<code>euporie.convert.formats.pdf</code>	Contains function which convert data to pdf format.
<code>euporie.convert.formats.pil</code>	Contains functions which convert data to PIL format.
<code>euporie.convert.formats.png</code>	Contains functions which convert data to png format.
<code>euporie.convert.formats.rich</code>	Contains function which convert data to rich format.
<code>euporie.convert.formats.sixel</code>	Contains function which convert data to sixel format.

euporie.convert.formats.ansi

Contains functions which convert data to formatted ansi text.

Functions

<code>html_to_ansi_elinks</code> (data[, width, height, ...])	Converts HTML text to formatted ANSI using elinks .
<code>html_to_ansi_links</code> (data[, width, height, fg, bg])	Converts HTML text to formatted ANSI using links .
<code>html_to_ansi_lynx</code> (data[, width, height, fg, bg])	Converts HTML text to formatted ANSI using lynx .
<code>html_to_ansi_py_htmlparser</code> (data[, width, ...])	Convert HTML tables to ANSI text using HTMLParser.
<code>html_to_ansi_w3m</code> (data[, width, height, fg, bg])	Converts HTML text to formatted ANSI using w3m .
<code>image_to_ansi_catimg</code> (data[, cols, rows, fg, bg])	Converts image data to ANSI text using catimg .
<code>image_to_ansi_icat</code> (data[, cols, rows, fg, bg])	Converts image data to ANSI text using icat .
<code>image_to_ansi_jp2a</code> (data[, cols, rows, fg, bg])	Converts image data to ANSI text using jp2a .
<code>image_to_ansi_timg</code> (data[, cols, rows, fg, bg])	Converts image data to ANSI text using timg .
<code>image_to_ansi_tiv</code> (data[, cols, rows, fg, bg])	Converts image data to ANSI text using tiv .
<code>image_to_ansi_viu</code> (data[, cols, rows, fg, bg])	Converts image data to ANSI text using viu .
<code>latex_to_ansi_py_flatlatex</code> (data[, width, ...])	Convert LaTeX to ANSI using flatlatex .
<code>latex_to_ansi_py_pylatexenc</code> (data[, width, ...])	Convert LaTeX to ANSI using flatlatex .
<code>latex_to_ansi_py_sympy</code> (data[, width, ...])	Convert LaTeX to ANSI using sympy .
<code>markdown_to_rich_py</code> (data[, width, height, ...])	Converts base64 encoded data to bytes.
<code>pil_to_ansi_py_img2unicode</code> (data[, cols, ...])	Convert a PIL image to ANSI text using timg .

continues on next page

Table 33 – continued from previous page

<code>pil_to_ansi_py_timg</code> (data[, cols, rows, fg, bg])	Convert a PIL image to ANSI text using <code>timg</code> .
<code>png_to_ansi_img2txt</code> (data[, cols, rows, fg, bg])	Converts PNG data to ANSI text using <code>img2txt</code> .
<code>png_to_ansi_py_placeholder</code> (data[, cols, ...])	Draw placeholder ANSI text.

euporie.convert.formats.ansi.html_to_ansi_elinks

`euporie.convert.formats.ansi.html_to_ansi_elinks`(data: *str*, width: *Optional[int]* = None, height: *Optional[int]* = None, fg: *Optional[str]* = None, bg: *Optional[str]* = None) → *str*

Converts HTML text to formatted ANSI using **elinks**.

euporie.convert.formats.ansi.html_to_ansi_links

`euporie.convert.formats.ansi.html_to_ansi_links`(data: *str*, width: *Optional[int]* = None, height: *Optional[int]* = None, fg: *Optional[str]* = None, bg: *Optional[str]* = None) → *str*

Converts HTML text to formatted ANSI using **links**.

euporie.convert.formats.ansi.html_to_ansi_lynx

`euporie.convert.formats.ansi.html_to_ansi_lynx`(data: *str*, width: *Optional[int]* = None, height: *Optional[int]* = None, fg: *Optional[str]* = None, bg: *Optional[str]* = None) → *str*

Converts HTML text to formatted ANSI using **lynx**.

euporie.convert.formats.ansi.html_to_ansi_py_htmlparser

`euporie.convert.formats.ansi.html_to_ansi_py_htmlparser`(data: *str*, width: *Optional[int]* = None, height: *Optional[int]* = None, fg: *Optional[str]* = None, bg: *Optional[str]* = None) → *str*

Convert HTML tables to ANSI text using `HTMLParser`.

euporie.convert.formats.ansi.html_to_ansi_w3m

`euporie.convert.formats.ansi.html_to_ansi_w3m`(data: *str*, width: *Optional[int]* = None, height: *Optional[int]* = None, fg: *Optional[str]* = None, bg: *Optional[str]* = None) → *str*

Converts HTML text to formatted ANSI using **w3m**.

euporie.convert.formats.ansi.image_to_ansi_catimg

`euporie.convert.formats.ansi.image_to_ansi_catimg`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **catimg**.

euporie.convert.formats.ansi.image_to_ansi_icat

`euporie.convert.formats.ansi.image_to_ansi_icat`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **icat**.

euporie.convert.formats.ansi.image_to_ansi_jp2a

`euporie.convert.formats.ansi.image_to_ansi_jp2a`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **jp2a**.

euporie.convert.formats.ansi.image_to_ansi_timg

`euporie.convert.formats.ansi.image_to_ansi_timg`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **timg**.

euporie.convert.formats.ansi.image_to_ansi_tiv

`euporie.convert.formats.ansi.image_to_ansi_tiv`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **tiv**.

euporie.convert.formats.ansi.image_to_ansi_viu

`euporie.convert.formats.ansi.image_to_ansi_viu`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **viu**.

euporie.convert.formats.ansi.latex_to_ansi_py_flatlatex

```
euporie.convert.formats.ansi.latex_to_ansi_py_flatlatex(data: str, width: Optional[int] = None,
                                                         height: Optional[int] = None, fg:
                                                         Optional[str] = None, bg: Optional[str] =
                                                         None) → str
```

Convert LaTeX to ANSI using flatlatex.

euporie.convert.formats.ansi.latex_to_ansi_py_pylatexenc

```
euporie.convert.formats.ansi.latex_to_ansi_py_pylatexenc(data: str, width: Optional[int] = None,
                                                           height: Optional[int] = None, fg:
                                                           Optional[str] = None, bg: Optional[str] =
                                                           None) → str
```

Convert LaTeX to ANSI using flatlatex.

euporie.convert.formats.ansi.latex_to_ansi_py_sympy

```
euporie.convert.formats.ansi.latex_to_ansi_py_sympy(data: str, width: Optional[int] = None, height:
                                                       Optional[int] = None, fg: Optional[str] = None,
                                                       bg: Optional[str] = None) → str
```

Convert LaTeX to ANSI using sympy.

euporie.convert.formats.ansi.markdown_to_rich_py

```
euporie.convert.formats.ansi.markdown_to_rich_py(data: RenderableType, width: Optional[int] = None,
                                                    height: Optional[int] = None, fg: Optional[str] =
                                                    None, bg: Optional[str] = None) → str
```

Converts base64 encoded data to bytes.

euporie.convert.formats.ansi.pil_to_ansi_py_img2unicode

```
euporie.convert.formats.ansi.pil_to_ansi_py_img2unicode(data: Image, cols: Optional[int] = None,
                                                           rows: Optional[int] = None, fg:
                                                           Optional[str] = None, bg: Optional[str] =
                                                           None) → str
```

Convert a PIL image to ANSI text using timimg.

euporie.convert.formats.ansi.pil_to_ansi_py_timimg

```
euporie.convert.formats.ansi.pil_to_ansi_py_timimg(data: Image, cols: Optional[int] = None, rows:
                                                       Optional[int] = None, fg: Optional[str] = None, bg:
                                                       Optional[str] = None) → str
```

Convert a PIL image to ANSI text using timimg.

euporie.convert.formats.ansi.png_to_ansi_img2txt

`euporie.convert.formats.ansi.png_to_ansi_img2txt`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts PNG data to ANSI text using **img2txt**.

euporie.convert.formats.ansi.png_to_ansi_py_placeholder

`euporie.convert.formats.ansi.png_to_ansi_py_placeholder`(data: *bytes*, cols: *int = 7*, rows: *int = 3*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Draw placeholder ANSI text.

`euporie.convert.formats.ansi.html_to_ansi_elinks`(data: *str*, width: *Optional[int] = None*, height: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts HTML text to formatted ANSI using **elinks**.

`euporie.convert.formats.ansi.html_to_ansi_links`(data: *str*, width: *Optional[int] = None*, height: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts HTML text to formatted ANSI using **links**.

`euporie.convert.formats.ansi.html_to_ansi_lynx`(data: *str*, width: *Optional[int] = None*, height: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts HTML text to formatted ANSI using **lynx**.

`euporie.convert.formats.ansi.html_to_ansi_py_htmlparser`(data: *str*, width: *Optional[int] = None*, height: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Convert HTML tables to ANSI text using **HTMLParser**.

`euporie.convert.formats.ansi.html_to_ansi_w3m`(data: *str*, width: *Optional[int] = None*, height: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts HTML text to formatted ANSI using **w3m**.

`euporie.convert.formats.ansi.image_to_ansi_catimg`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **catimg**.

`euporie.convert.formats.ansi.image_to_ansi_icat`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **icat**.

`euporie.convert.formats.ansi.image_to_ansi_jp2a`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **jp2a**.

`euporie.convert.formats.ansi.image_to_ansi_timg`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **timg**.

`euporie.convert.formats.ansi.image_to_ansi_tiv`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **tiv**.

`euporie.convert.formats.ansi.image_to_ansi_viu`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts image data to ANSI text using **viu**.

`euporie.convert.formats.ansi.latex_to_ansi_py_flatlatex`(data: *str*, width: *Optional[int] = None*, height: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Convert LaTeX to ANSI using flatlatex.

`euporie.convert.formats.ansi.latex_to_ansi_py_pylatexenc`(data: *str*, width: *Optional[int] = None*, height: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Convert LaTeX to ANSI using flatlatex.

`euporie.convert.formats.ansi.latex_to_ansi_py_sympy`(data: *str*, width: *Optional[int] = None*, height: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Convert LaTeX to ANSI using **sympy**.

`euporie.convert.formats.ansi.markdown_to_rich_py`(data: *RenderableType*, width: *Optional[int] = None*, height: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts base64 encoded data to bytes.

`euporie.convert.formats.ansi.pil_to_ansi_py_img2unicode`(data: *Image*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Convert a PIL image to ANSI text using **timg**.

`euporie.convert.formats.ansi.pil_to_ansi_py_timg`(data: *Image*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Convert a PIL image to ANSI text using **timg**.

`euporie.convert.formats.ansi.png_to_ansi_img2txt`(data: *bytes*, cols: *Optional[int] = None*, rows: *Optional[int] = None*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Converts PNG data to ANSI text using **img2txt**.

`euporie.convert.formats.ansi.png_to_ansi_py_placeholder`(data: *bytes*, cols: *int = 7*, rows: *int = 3*, fg: *Optional[str] = None*, bg: *Optional[str] = None*) → *str*

Draw placeholder ANSI text.

euporie.convert.formats.base64

Contains functions which convert data to base64 format.

Functions

<code>bytes_to_base64_py(data[, width, height, fg, bg])</code>	Converts bytes to base64 encoded data.
--	--

euporie.convert.formats.base64.bytes_to_base64_py

```
euporie.convert.formats.base64.bytes_to_base64_py(data: Union[str, bytes], width: Optional[int] =  
None, height: Optional[int] = None, fg:  
Optional[str] = None, bg: Optional[str] = None)  
→ str
```

Converts bytes to base64 encoded data.

```
euporie.convert.formats.base64.bytes_to_base64_py(data: Union[str, bytes], width: Optional[int] =  
None, height: Optional[int] = None, fg:  
Optional[str] = None, bg: Optional[str] = None)  
→ str
```

Converts bytes to base64 encoded data.

euporie.convert.formats.common

Contains functions which can be used to convert data to multiple formats.

Functions

<code>base64_to_bytes_py(data[, width, height, fg, bg])</code>	Converts base64 encoded data to bytes.
<code>chafa_convert(output_format, data[, cols, ...])</code>	Converts image data to ANSI text using chafa .
<code>imagemagick_convert(output_format, data[, ...])</code>	Converts image data to PNG bytes using imagemagick .

euporie.convert.formats.common.base64_to_bytes_py

```
euporie.convert.formats.common.base64_to_bytes_py(data: str, width: Optional[int] = None, height:  
Optional[int] = None, fg: Optional[str] = None,  
bg: Optional[str] = None) → bytes
```

Converts base64 encoded data to bytes.

euporie.convert.formats.common.chafa_convert

`euporie.convert.formats.common.chafa_convert`(*output_format*: *str*, *data*: *Union[bytes, str]*, *cols*: *Optional[int]* = *None*, *rows*: *Optional[int]* = *None*, *fg*: *Optional[str]* = *None*, *bg*: *Optional[str]* = *None*) → *Union[str, bytes]*

Converts image data to ANSI text using **chafa**.

euporie.convert.formats.common.imagemagick_convert

`euporie.convert.formats.common.imagemagick_convert`(*output_format*: *str*, *data*: *Union[str, bytes]*, *cols*: *Optional[int]* = *None*, *rows*: *Optional[int]* = *None*, *fg*: *Optional[str]* = *None*, *bg*: *Optional[str]* = *None*) → *Union[str, bytes]*

Converts image data to PNG bytes using **imagemagick**.

`euporie.convert.formats.common.base64_to_bytes_py`(*data*: *str*, *width*: *Optional[int]* = *None*, *height*: *Optional[int]* = *None*, *fg*: *Optional[str]* = *None*, *bg*: *Optional[str]* = *None*) → *bytes*

Converts base64 encoded data to bytes.

`euporie.convert.formats.common.chafa_convert`(*output_format*: *str*, *data*: *Union[bytes, str]*, *cols*: *Optional[int]* = *None*, *rows*: *Optional[int]* = *None*, *fg*: *Optional[str]* = *None*, *bg*: *Optional[str]* = *None*) → *Union[str, bytes]*

Converts image data to ANSI text using **chafa**.

`euporie.convert.formats.common.imagemagick_convert`(*output_format*: *str*, *data*: *Union[str, bytes]*, *cols*: *Optional[int]* = *None*, *rows*: *Optional[int]* = *None*, *fg*: *Optional[str]* = *None*, *bg*: *Optional[str]* = *None*) → *Union[str, bytes]*

Converts image data to PNG bytes using **imagemagick**.

euporie.convert.formats.jpeg

Contains functions which convert data to jpeg format.

euporie.convert.formats.markdown

Contains functions which convert data to markdown format.

Functions

<code>html_to_markdown_py_mtable</code> (<i>data</i> [, <i>width</i> , ...])	Convert HTML tables to markdown tables using <code>mtable</code> .
---	--

euporie.convert.formats.markdown.html_to_markdown_py_mtable

`euporie.convert.formats.markdown.html_to_markdown_py_mtable`(data: *str*, width: *Optional[int]* = None, height: *Optional[int]* = None, fg: *Optional[str]* = None, bg: *Optional[str]* = None) → *str*

Convert HTML tables to markdown tables using mtable.

`euporie.convert.formats.markdown.html_to_markdown_py_mtable`(data: *str*, width: *Optional[int]* = None, height: *Optional[int]* = None, fg: *Optional[str]* = None, bg: *Optional[str]* = None) → *str*

Convert HTML tables to markdown tables using mtable.

euporie.convert.formats.pdf

Contains function which convert data to pdf format.

euporie.convert.formats.pil

Contains functions which convert data to PIL format.

Functions

<code>png_to_pil_py</code> (data[, cols, rows, fg, bg])	Convert PNG to a pillow image using PIL.
<code>set_background</code> (image[, bg_color])	Removes the alpha channel from an image and set the background colour.

euporie.convert.formats.pil.png_to_pil_py

`euporie.convert.formats.pil.png_to_pil_py`(data: *bytes*, cols: *Optional[int]* = None, rows: *Optional[int]* = None, fg: *Optional[str]* = None, bg: *Optional[str]* = None) → *Image*

Convert PNG to a pillow image using PIL.

euporie.convert.formats.pil.set_background

`euporie.convert.formats.pil.set_background`(image: *Image*, bg_color: *Optional[str]* = None) → *bytes*
Removes the alpha channel from an image and set the background colour.

`euporie.convert.formats.pil.png_to_pil_py`(data: *bytes*, cols: *Optional[int]* = None, rows: *Optional[int]* = None, fg: *Optional[str]* = None, bg: *Optional[str]* = None) → *Image*

Convert PNG to a pillow image using PIL.

`euporie.convert.formats.pil.set_background`(image: *Image*, bg_color: *Optional[str]* = None) → *bytes*
Removes the alpha channel from an image and set the background colour.

euporie.convert.formats.png

Contains functions which convert data to png format.

Functions

<code>latex_to_png_py_ipython(backend, data[, ...])</code>	Converts LaTeX data to PNG bytes with IPython & matplotlib.
<code>pil_to_png_py_pil(data[, cols, rows, fg, bg])</code>	Convert a pillow image to sixels teimpy.
<code>svg_to_png_py_cairosvg(data[, width, ...])</code>	Convert SVG to PNG using cairosvg.

euporie.convert.formats.png.latex_to_png_py_ipython

`euporie.convert.formats.png.latex_to_png_py_ipython(backend: str, data: str, width: Optional[int] = None, height: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → bytes`

Converts LaTeX data to PNG bytes with IPython & matplotlib.

euporie.convert.formats.png.pil_to_png_py_pil

`euporie.convert.formats.png.pil_to_png_py_pil(data: Image, cols: Optional[int] = None, rows: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → bytes`

Convert a pillow image to sixels teimpy.

euporie.convert.formats.png.svg_to_png_py_cairosvg

`euporie.convert.formats.png.svg_to_png_py_cairosvg(data: str, width: Optional[int] = None, height: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → str`

Convert SVG to PNG using cairosvg.

`euporie.convert.formats.png.latex_to_png_py_ipython(backend: str, data: str, width: Optional[int] = None, height: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → bytes`

Converts LaTeX data to PNG bytes with IPython & matplotlib.

`euporie.convert.formats.png.pil_to_png_py_pil(data: Image, cols: Optional[int] = None, rows: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → bytes`

Convert a pillow image to sixels teimpy.

`euporie.convert.formats.png.svg_to_png_py_cairosvg(data: str, width: Optional[int] = None, height: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → str`

Convert SVG to PNG using cairosvg.

euporie.convert.formats.rich

Contains function which convert data to rich format.

Functions

<code>markdown_to_rich_py(data[, width, height, ...])</code>	Converts base64 encoded data to bytes.
--	--

euporie.convert.formats.rich.markdown_to_rich_py

`euporie.convert.formats.rich.markdown_to_rich_py(data: str, width: Optional[int] = None, height: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → Markdown`

Converts base64 encoded data to bytes.

`euporie.convert.formats.rich.markdown_to_rich_py(data: str, width: Optional[int] = None, height: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → Markdown`

Converts base64 encoded data to bytes.

euporie.convert.formats.sixel

Contains function which convert data to sixel format.

Functions

<code>pil_to_sixel_py_timgpy(data[, cols, rows, ...])</code>	Convert a pillow image to sixels timgpy .
<code>pil_to_sixel_py_timg(data[, cols, rows, fg, bg])</code>	Convert a pillow image to sixels timg .
<code>png_to_sixel_img2sixel(data[, cols, rows, ...])</code>	Converts PNG data to sixels img2sixel .

euporie.convert.formats.sixel.pil_to_sixel_py_timgpy

`euporie.convert.formats.sixel.pil_to_sixel_py_timgpy(data: Image, cols: Optional[int] = None, rows: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → str`

Convert a pillow image to sixels timgpy.

euporie.convert.formats.sixel.pil_to_sixel_py_timg

`euporie.convert.formats.sixel.pil_to_sixel_py_timg`(*data: Image, cols: Optional[int] = None, rows: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None*) → *str*

Convert a pillow image to sixels timg.

euporie.convert.formats.sixel.png_to_sixel_img2sixel

`euporie.convert.formats.sixel.png_to_sixel_img2sixel`(*data: bytes, cols: Optional[int] = None, rows: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None*) → *str*

Converts PNG data to sixels **img2sixel**.

`euporie.convert.formats.sixel.pil_to_sixel_py_timg`(*data: Image, cols: Optional[int] = None, rows: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None*) → *str*

Convert a pillow image to sixels teimpy.

`euporie.convert.formats.sixel.pil_to_sixel_py_timg`(*data: Image, cols: Optional[int] = None, rows: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None*) → *str*

Convert a pillow image to sixels timg.

`euporie.convert.formats.sixel.png_to_sixel_img2sixel`(*data: bytes, cols: Optional[int] = None, rows: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None*) → *str*

Converts PNG data to sixels **img2sixel**.

euporie.convert.util

Utility functions for format convertors.

Functions

<code>call_subproc</code> (<i>data, cmd[, use_tempfile]</i>)	Call the command as a subprocess and return it's output as bytes.
<code>commands_exist</code> (* <i>cmds</i>)	Verifies a list of external commands exist on the system.
<code>have_modules</code> (* <i>modules</i>)	Verifies a list of python modules are importable.

euporie.convert.util.call_subproc

`euporie.convert.util.call_subproc`(*data: Union[str, bytes], cmd: list[Any], use_tempfile: bool = False*) → *bytes*

Call the command as a subprocess and return it's output as bytes.

Parameters

- **data** – The data to pass to the subprocess
- **cmd** – The command and arguments to call

- **use_tempfile** – If True, the command saves its output to a file, not stdout

Returns The data printed to standard out by the subprocess.

euporie.convert.util.commands_exist

euporie.convert.util.commands_exist(*cmds: str) → Filter
Verifies a list of external commands exist on the system.

euporie.convert.util.have_modules

euporie.convert.util.have_modules(*modules: str) → Filter
Verifies a list of python modules are importable.

euporie.convert.util.call_subproc(data: Union[str, bytes], cmd: list[Any], use_tempfile: bool = False) → bytes
Call the command as a subprocess and return it's output as bytes.

Parameters

- **data** – The data to pass to the subprocess
- **cmd** – The command and arguments to call
- **use_tempfile** – If True, the command saves its output to a file, not stdout

Returns The data printed to standard out by the subprocess.

euporie.convert.util.commands_exist(*cmds: str) → Filter
Verifies a list of external commands exist on the system.

euporie.convert.util.have_modules(*modules: str) → Filter
Verifies a list of python modules are importable.

5.8.9 euporie.filters

Defines common filters.

5.8.10 euporie.format

Contains functions to automatically format code cell input.

Functions

<code>format_black(text)</code>	Format a code string using black.
<code>format_code(text)</code>	Format a code string using :py:mod:``.
<code>format_isort(text)</code>	Format a code string using isort.
<code>format_ssort(text)</code>	Format a code string using ssort.

euporie.format.format_black

`euporie.format.format_black(text: str) → str`
Format a code string using black.

euporie.format.format_code

`euporie.format.format_code(text: str) → str`
Format a code string using `:py:mod:```.

euporie.format.format_isort

`euporie.format.format_isort(text: str) → str`
Format a code string using isort.

euporie.format.format_ssort

`euporie.format.format_ssort(text: str) → str`
Format a code string using ssort.

`euporie.format.format_black(text: str) → str`
Format a code string using black.

`euporie.format.format_code(text: str) → str`
Format a code string using `:py:mod:```.

`euporie.format.format_isort(text: str) → str`
Format a code string using isort.

`euporie.format.format_ssort(text: str) → str`
Format a code string using ssort.

5.8.11 euporie.kernel

Contains the main class for a notebook file.

Classes

<code>NotebookKernel(name[, threaded, allow_stdin])</code>	Runs a notebook kernel and communicates with it asynchronously.
--	---

euporie.kernel.NotebookKernel

class euporie.kernel.**NotebookKernel**(name: *str*, threaded: *bool* = True, allow_stdin: *bool* = False)
Runs a notebook kernel and communicates with it asynchronously.

Has the ability to run itself in it's own thread.

class euporie.kernel.**NotebookKernel**(name: *str*, threaded: *bool* = True, allow_stdin: *bool* = False)
Bases: `object`

Runs a notebook kernel and communicates with it asynchronously.

Has the ability to run itself in it's own thread.

async **await_iopub_rsps**(msg_id: *str*) → jupyter_client.manager.AsyncKernelManager
Wait for messages on the `iopub` channel.

This will yield response message, stopping after a response with a status value of “idle” or a type of “error”.

Parameters `msg_id` – The ID of the message to process the responses for.

Yields The response message

async **await_rsps**(msg_id: *str*, channel: *str*, timeout: *int* = 360) → AsyncGenerator
Yields responses to a given message ID on a given channel.

Parameters

- `msg_id` – Wait for responses to this message ID
- `channel` – The channel to listen on for responses
- `timeout` – Maximum time to wait for a response before stopping

Yields Message responses received on the given channel

async **await_shell_rsps**(msg_id: *str*) → jupyter_client.manager.AsyncKernelManager
Wait for messages on the `shell` channel.

This will yield response message, stopping after a response with a status of “ok” or “error”.

Parameters `msg_id` – The ID of the message to process the responses for.

Yields The response message

async **await_stdin_rsps**(msg_id: *str*) → jupyter_client.manager.AsyncKernelManager
Wait for messages on the `shell` channel.

This will yield response message, stopping after a response with a status of “ok” or “error”.

Parameters `msg_id` – The ID of the message to process the responses for.

Yields The response message

change(name: *str*, metadata_json: *dict*) → `None`
Change the kernel.

Parameters

- `name` – The name of the kernel to change to
- `metadata_json` – The notebook's metadata, so the kernel notebook's kernelspec metadata can be updated

complete(code: *str*, cursor_pos: *int*) → `list[dict]`
Request code completions from the kernel.

Parameters

- **code** – The code string to retrieve completions for
- **cursor_pos** – The position of the cursor in the code string

Returns A list of dictionaries defining completion entries. The dictionaries contain **text** (the completion text), **start_position** (the starting position of the completion text), and optionally **display_meta** (a string containing additional data about the completion type)

async complete_(*code: str, cursor_pos: int*) → list[dict]

Request code completions from the kernel, asynchronously.

history(*pattern: str, n: int = 1*) → Optional[list[tuple[int, int, str]]]

Retrieve history from the kernel.

Parameters

- **pattern** – The pattern to search for
- **n** – the number of history items to return

Returns A list of history items, consisting of tuples (session, line_number, input)

async history_(*pattern: str, n: int = 1*) → Optional[list[tuple[int, int, str]]]

Retrieve history from the kernel asynchronously.

property id: Optional[str]

Get the ID of the current kernel.

info(*cb: Optional[Callable[[dict], Any]] = None, wait: bool = False*) → dict

Request information about the kernel.

Parameters

- **cb** – A callback function to run when the information has been retrieved. The info is passed as an argument
- **wait** – If True, block the main thread until complete

Returns The kernel info

async info_() → dict

Request information about the kernel.

Returns The kernel info

inspect(*code: str, cursor_pos: int, callback: Callable[[dict[str, Any]], None] = None*) → str

Request code inspection from the kernel.

Parameters

- **code** – The code string to retrieve completions for
- **cursor_pos** – The position of the cursor in the code string
- **callback** – A function to run when the inspection result arrives. The result is passed as an argument.

Returns A string containing useful information about the code at the current cursor position

async inspect_(*code: str, cursor_pos: int, detail_level: int = 0*) → dict[str, Any]

Retrieve introspection string from the kernel asynchronously.

interrupt() → None

Interrupt the kernel.

This is run in the main thread rather than on the event loop in the kernel's thread, because otherwise we would have to wait for currently running tasks on the kernel's event loop to finish.

property missing: `bool`

Returns a list of available kernelspecs.

async poll(*channel: str*) → `None`

Polls for messages on a channel, and signal when they arrive.

Parameters `channel` – The name of the channel to get messages from

async process_default_iopub_rsp(*msg_id: str*) → `None`

The default processor for message responses on the `iopub` channel.

This does nothing when a response is received.

Parameters `msg_id` – The ID of the message to process the responses for.

restart(*wait: bool = False*) → `None`

Restarts the current kernel.

async restart_() → `None`

Restart the kernel asynchronously.

run(*cell_json: dict, stdin_cb: Optional[Callable[... Any]] = None, output_cb: Optional[Callable[[], Any]] = None, done_cb: Optional[Callable[[], Any]] = None, wait: bool = False*) → `None`

Run a cell using the notebook kernel and process the responses.

Cell output is added to the cell json.

Parameters

- **cell_json** – The JSON representation of the cell to run
- **stdin_cb** – An optional coroutine callback to run when the kernel requests input. Should accept a function which should be called with the user input as the only argument
- **output_cb** – An optional callback to run after each response message
- **done_cb** – An optional callback to run when the cell has finished running
- **wait** – If `:py:const`True``, will block until the cell has finished running

async run_(*cell_json: dict, stdin_cb: Optional[Callable[... Any]] = None, output_cb: Optional[Callable[[], Any]] = None, done_cb: Optional[Callable[[], Any]] = None*) → `None`

Runs the code cell asynchronously and handles the responses.

set_metadata(*cell_json: dict, path: tuple[str, ...], data: Any*) → `None`

Sets a value in the metadata at an arbitrary path.

Parameters

- **cell_json** – The cell_json to add the meta data to
- **path** – A tuple of path level names to create
- **data** – The value to add

shutdown(*wait: bool = False*) → `None`

Shutdown the kernel and close the kernel's thread.

This is intended to be run when the notebook is closed: the `NotebookKernel` cannot be restarted after this.

Parameters `wait` – Whether to block until shutdown completes

async shutdown_() → `None`

Shut down the kernel and close the event loop if running in a thread.

property specs: `dict[str, dict]`

Returns a list of available kernelspecs.

start(*cb: Optional[Callable] = None, wait: bool = False, timeout: int = 10*) → `None`

Starts the kernel.

Parameters

- **cb** – An optional callback to run after the kernel has started
- **wait** – If `True`, block until the kernel has started
- **timeout** – How long to wait until failure is assumed

async start_*(cb: Optional[Callable] = None)* → `None`

Start the kernel asynchronously and set its status.

property status: `str`

Retrieve the current kernel status.

Trigger a kernel life status check when retrieved

Returns The kernel status

stop(*cb: Optional[Callable] = None, wait: bool = False*) → `None`

Stops the current kernel.

Parameters

- **cb** – An optional callback to run when the kernel has stopped.
- **wait** – If `True`, wait for the kernel to become idle, otherwise the kernel is interrupted before it is stopped

async stop_*(cb: Optional[Callable[[], Any]] = None)* → `None`

Stop the kernel asynchronously.

5.8.12 euporie.key_binding

Defines key-bindings for the application.

Modules

<code>euporie.key_binding.bindings</code>	Defines sets of key-bindings.
<code>euporie.key_binding.micro_state</code>	Defines the state of the micro editing mode.
<code>euporie.key_binding.util</code>	Utility functions for loading key-bindings.

euporie.key_binding.bindings

Defines sets of key-bindings.

Modules

<code>euporie.key_binding.bindings.commands</code>	Load the default bindings assigned to commands.
<code>euporie.key_binding.bindings.micro</code>	Defines editor key-bindings in the style of the <code>micro</code> text editor.

euporie.key_binding.bindings.commands

Load the default bindings assigned to commands.

Functions

<code>load_command_bindings(*groups)</code>	Loads key-bindings for command belonging to a list of groups.
---	---

euporie.key_binding.bindings.commands.load_command_bindings

`euporie.key_binding.bindings.commands.load_command_bindings(*groups: Optional[str]) → KeyBindingsBase`

Loads key-bindings for command belonging to a list of groups.

`euporie.key_binding.bindings.commands.load_command_bindings(*groups: Optional[str]) → KeyBindingsBase`

Loads key-bindings for command belonging to a list of groups.

euporie.key_binding.bindings.micro

Defines editor key-bindings in the style of the `micro` text editor.

Functions

<code>load_micro_bindings()</code>	Load editor key-bindings in the style of the <code>micro</code> text editor.
------------------------------------	--

euporie.key_binding.bindings.micro.load_micro_bindings

`euporie.key_binding.bindings.micro.load_micro_bindings()` → `KeyBindingsBase`
 Load editor key-bindings in the style of the micro text editor.

`euporie.key_binding.bindings.micro.load_micro_bindings()` → `KeyBindingsBase`
 Load editor key-bindings in the style of the micro text editor.

euporie.key_binding.micro_state

Defines the state of the micro editing mode.

Classes

<code>InputMode(value)</code>	Enum to define edit mode state types.
<code>MicroState()</code>	Mutable class to hold Micro specific state.

euporie.key_binding.micro_state.InputMode

class `euporie.key_binding.micro_state.InputMode(value)`
 Enum to define edit mode state types.

euporie.key_binding.micro_state.MicroState

class `euporie.key_binding.micro_state.MicroState`
 Mutable class to hold Micro specific state.

class `euporie.key_binding.micro_state.InputMode(value)`
 Bases: `str`, `enum.Enum`

Enum to define edit mode state types.

INSERT = 'insert'

REPLACE = 'replace'

value: `str`

class `euporie.key_binding.micro_state.MicroState`
 Bases: `object`

Mutable class to hold Micro specific state.

end_macro() → `None`
 End recording a macro.

property is_recording: `bool`
 Tell whether we are recording a macro.

reset() → `None`
 Reset the editing mode state.

start_macro() → `None`
 Start recording a macro.

euporie.key_binding.util

Utility functions for loading key-bindings.

Functions

<code>dict_bindings(binding_dict)</code>	Assign key-bindings to commands based on a dictionary.
<code>format_keys(keys)</code>	Converts a list of tuples of keys to a string.

euporie.key_binding.util.dict_bindings

`euporie.key_binding.util.dict_bindings(binding_dict: Dict[str, AnyKeys]) → KeyBindingsBase`
Assign key-bindings to commands based on a dictionary.

euporie.key_binding.util.format_keys

`euporie.key_binding.util.format_keys(keys: List[Tuple[Union[str, Keys], ...]]) → List[str]`
Converts a list of tuples of keys to a string.

`euporie.key_binding.util.dict_bindings(binding_dict: Dict[str, AnyKeys]) → KeyBindingsBase`
Assign key-bindings to commands based on a dictionary.

5.8.13 euporie.keys

Defines KeyBindings wrapper which keeps track of key binding descriptions.

Classes

<code>KeyBindingsInfo()</code>	Wraps <code>prompt_toolkit.key_binding.KeyBinding</code> recording helpful details.
--------------------------------	---

euporie.keys.KeyBindingsInfo

class `euporie.keys.KeyBindingsInfo`

Wraps `prompt_toolkit.key_binding.KeyBinding` recording helpful details.

Each key binding can be given a group and a description, which can be used to display a help message about which key bindings are available to the user.

class `euporie.keys.KeyBindingsInfo`

Bases: `prompt_toolkit.key_binding.key_bindings.KeyBindings`

Wraps `prompt_toolkit.key_binding.KeyBinding` recording helpful details.

Each key binding can be given a group and a description, which can be used to display a help message about which key bindings are available to the user.

add(**keys: Union[Keys, str]*, *key_str: Optional[tuple[str]] = None*, *group: str = 'None'*, *desc: Optional[str] = None*, ***kwargs: Any*) → Callable[[T], T]
Decorator for adding a key bindings.

As per `prompt_toolkit.key_binding.KeyBinding`, with additional arguments.

Parameters

- ***keys** – Keys to pass to `prompt_toolkit.key_binding.KeyBinding.add`.
- **key_str** – A string which can be user to over-ride the bound key sequence in the binding's description.
- **group** – The name of the group to which this key binding belongs.
- **desc** – A description of what this key-binding does.
- ****kwargs** – Key word arguments to pass to `prompt_toolkit.key_binding.KeyBinding.add`.

Returns A decorator function.

```
add_binding(*keys: typing.Union[prompt_toolkit.keys.Keys, str], filter:
    typing.Union[prompt_toolkit.filters.base.Filter, bool] = True, eager:
    typing.Union[prompt_toolkit.filters.base.Filter, bool] = False, is_global:
    typing.Union[prompt_toolkit.filters.base.Filter, bool] = False, save_before:
    typing.Callable[[KeyPressEvent], bool] = <function KeyBindings.<lambda>>,
    record_in_macro: typing.Union[prompt_toolkit.filters.base.Filter, bool] = True) →
    Callable[[prompt_toolkit.key_binding.key_bindings.T],
    prompt_toolkit.key_binding.key_bindings.T]
```

Decorator for adding a key bindings.

Parameters

- **filter** – `Filter` to determine when this key binding is active.
- **eager** – `Filter` or `bool`. When True, ignore potential longer matches when this key binding is hit. E.g. when there is an active eager key binding for Ctrl-X, execute the handler immediately and ignore the key binding for Ctrl-X Ctrl-E of which it is a prefix.
- **is_global** – When this key bindings is added to a *Container* or *Control*, make it a global (always active) binding.
- **save_before** – Callable that takes an *Event* and returns True if we should save the current buffer, before handling the event. (That's the default.)
- **record_in_macro** – Record these key bindings when a macro is being recorded. (True by default.)

property bindings: `List[prompt_toolkit.key_binding.key_bindings.Binding]`

List of *Binding* objects. (These need to be exposed, so that *KeyBindings* objects can be merged together.)

details: `dict[str, dict[str, dict[tuple[Union[Keys, str]], None]]] = {}`

get_bindings_for_keys(keys: `Tuple[Union[prompt_toolkit.keys.Keys, str], ...]`) →

`List[prompt_toolkit.key_binding.key_bindings.Binding]`

Return a list of key bindings that can handle this key. (This return also inactive bindings, so the *filter* still has to be called, for checking it.)

Parameters **keys** – tuple of keys.

get_bindings_starting_with_keys(keys: `Tuple[Union[prompt_toolkit.keys.Keys, str], ...]`) →

`List[prompt_toolkit.key_binding.key_bindings.Binding]`

Return a list of key bindings that handle a key sequence starting with *keys*. (It does only return bindings for which the sequences are longer than *keys*. And like *get_bindings_for_keys*, it also includes inactive bindings.)

Parameters **keys** – tuple of keys.

```
remove(*args: Union[prompt_toolkit.keys.Keys, str, Callable[[KeyPressEvent],  
    Union[NotImplementedOrNone, Awaitable[NotImplementedOrNone]]]]) → None
```

Remove a key binding.

This expects either a function that was given to *add* method as parameter or a sequence of key bindings.

Raises *ValueError* when no bindings was found.

Usage:

```
remove(handler) # Pass handler.  
remove('c-x', 'c-a') # Or pass the key bindings.
```

```
remove_binding(*args: Union[prompt_toolkit.keys.Keys, str, Callable[[KeyPressEvent],  
    Union[NotImplementedOrNone, Awaitable[NotImplementedOrNone]]]]) → None
```

Remove a key binding.

This expects either a function that was given to *add* method as parameter or a sequence of key bindings.

Raises *ValueError* when no bindings was found.

Usage:

```
remove(handler) # Pass handler.  
remove('c-x', 'c-a') # Or pass the key bindings.
```

```
classmethod to_formatted_text() → FormattedText
```

Format the current key binding descriptions as formatted text.

Returns Formatted text giving a description of the key-bindings in each group.

5.8.14 euporie.log

Initiate logging for euporie.

Functions

<i>setup_logs</i> ()	Configures the logger for euporie.
----------------------	------------------------------------

euporie.log.setup_logs

```
euporie.log.setup_logs() → None
```

Configures the logger for euporie.

Classes

<code>LogView()</code>	A tab which allows you to view log entries.
<code>QueueHandler(queue)</code>	This handler store logs events into a queue.
<code>stdout_to_log(log[, output])</code>	A decorator which captures standard output and logs it.

euporie.log.LogView

class euporie.log.LogView
A tab which allows you to view log entries.

euporie.log.QueueHandler

class euporie.log.QueueHandler(*queue: collections.deque*)
This handler store logs events into a queue.

euporie.log.stdout_to_log

class euporie.log.stdout_to_log(*log: logging.Logger, output: str = "Literal['stdout','stderr']"*)
A decorator which captures standard output and logs it.

class euporie.log.LogView
Bases: `euporie.tab.Tab`
A tab which allows you to view log entries.

add_record(*record: logging.LogRecord*) → None
Adds a single new record to the textarea.

Parameters **record** – The log record to add

close(*cb: Optional[Callable]*) → None
Function to close a tab with a callback.

Parameters **cb** – A function to call after the tab is closed.

container: `_Split`

focus() → None
Focuses the tab.

render(*record: logging.LogRecord*) → StyleAndTextTuples
Converts a log record to formatted text.

Parameters **record** – The log record to format

Returns A list of style and text tuples describing the log record

statusbar_fields() → tuple[Sequence[AnyFormattedText], Sequence[AnyFormattedText]]
Returns a list of statusbar field values shown then this tab is active.

property title: `str`
Returns the title of this tab.

class euporie.log.QueueHandler(*queue: collections.deque*)
Bases: `logging.Handler`
This handler store logs events into a queue.

acquire()

Acquire the I/O thread lock.

addFilter(*filter*)

Add the specified filter to this handler.

close()

Tidy up any resources used by the handler.

This version removes the handler from an internal map of handlers, `_handlers`, which is used for handler lookup by name. Subclasses should ensure that this gets called from overridden `close()` methods.

createLock()

Acquire a thread lock for serializing access to the underlying I/O.

emit(*record*: *logging.LogRecord*) → None

Queue unformatted records, as they will be formatted when accessed.

filter(*record*)

Determine if a record is loggable by consulting all the filters.

The default is to allow the record to be logged; any filter can veto this and the record is then dropped. Returns a zero value if a record is to be dropped, else non-zero.

Changed in version 3.2: Allow filters to be just callables.

flush()

Ensure all logging output has been flushed.

This version does nothing and is intended to be implemented by subclasses.

format(*record*)

Format the specified record.

If a formatter is set, use it. Otherwise, use the default formatter for the module.

get_name()**handle(*record*)**

Conditionally emit the specified logging record.

Emission depends on filters which may have been added to the handler. Wrap the actual emission of the record with acquisition/release of the I/O thread lock. Returns whether the filter passed the record for emission.

handleError(*record*)

Handle errors which occur during an `emit()` call.

This method should be called from handlers when an exception is encountered during an `emit()` call. If `raiseExceptions` is false, exceptions get silently ignored. This is what is mostly wanted for a logging system - most users will not care about errors in the logging system, they are more interested in application errors. You could, however, replace this with a custom handler if you wish. The record which was being processed is passed in to this method.

classmethod hook(*hook*: Callable) → int

Adds a hook to run after each log entry.

Parameters **hook** – The hook function to add

Returns The hook id

hook_id = 0

hooks: dict[int, Callable] = {}

property name**release()**

Release the I/O thread lock.

removeFilter(*filter*)

Remove the specified filter from this handler.

setFormatter(*fnt*)

Set the formatter for this handler.

setLevel(*level*)

Set the logging level of this handler. level must be an int or a str.

set_name(*name*)**classmethod unhook(*hook_id: int*) → None**

Removes a hook function.

Parameters **hook_id** – The ID of the hook function to remove

euporie.log.setup_logs() → None

Configures the logger for euporie.

5.8.15 euporie.margins

Contains margins.

Classes

<i>ScrollbarMargin</i> ([<i>display_arrows</i> , ...])	Margin displaying a scrollbar.
---	--------------------------------

euporie.margins.ScrollbarMargin

```
class euporie.margins.ScrollbarMargin(display_arrows: Union[prompt_toolkit.filters.base.Filter, bool] = True, up_arrow_symbol: str = "⬆️", down_arrow_symbol: str = "⬇️", smooth: bool = True)
```

Margin displaying a scrollbar.

Parameters

- **display_arrows** – Display scroll up/down arrows.
- **up_arrow** – Character to use for the scrollbar's up arrow
- **down_arrow** – Character to use for the scrollbar's down arrow
- **smooth** – Use block character to move scrollbar more smoothly

```
class euporie.margins.ScrollbarMargin(display_arrows: Union[prompt_toolkit.filters.base.Filter, bool] = True, up_arrow_symbol: str = "⬆️", down_arrow_symbol: str = "⬇️", smooth: bool = True)
```

Bases: `prompt_toolkit.layout.margins.Margin`

Margin displaying a scrollbar.

Parameters

- **display_arrows** – Display scroll up/down arrows.

- **up_arrow** – Character to use for the scrollbar’s up arrow
- **down_arrow** – Character to use for the scrollbar’s down arrow
- **smooth** – Use block character to move scrollbar more smoothly

create_margin(*window_render_info*: *WindowRenderInfo*, *width*: *int*, *height*: *int*) → *StyleAndTextTuples*
Creates the margin’s formatted text.

eighths = ' '

get_width(*get_ui_content*: *Callable[[], UIContent]*) → *int*
Return the scrollbar width: always 1.

mouse_handler(*mouse_event*: *MouseEvent*, *repeated*: *bool* = *False*) → *None*
Handle scrollbar mouse events.

Scrolls up or down if the arrows are clicked, repeating while the mouse button is held down. Scrolls up or down one page if the background is clicked, repeating while the left mouse button is held down. Scrolls if the scrollbar-button is dragged. Scrolls if the scroll-wheel is used on the scrollbar.

Parameters

- **mouse_event** – The triggering mouse event
- **repeated** – Set to True if the method is running as a repeated event

async repeat(*mouse_event*: *MouseEvent*, *timeout*: *float* = *0.1*) → *None*
Repeat a mouse event after a timeout.

window_render_info: **WindowRenderInfo**

5.8.16 euporie.markdown

Defines markdown extensions for commonmark and rich.

Modules

<code>euporie.markdown.blocks</code>	Defines the start of blocks for extensions to commonmark.
<code>euporie.markdown.inlines</code>	Defines additional inline nodes for the python commonmark parser.
<code>euporie.markdown.parser</code>	Adds extensions to the python commonmark parser.
<code>euporie.markdown.rich</code>	Adds extends <code>:py:class`rich.markdown.Markdown`</code> with tables and LaTeX.

euporie.markdown.blocks

Defines the start of blocks for extensions to commonmark.

Modules

<code>euporie.markdown.blocks.math</code>	A math block extensions for python commonmark.
<code>euporie.markdown.blocks.tables</code>	An extension for python commonmark to include markdown tables.

euporie.markdown.blocks.math

A math block extensions for python commonmark.

Classes

<code>MathBlock()</code>	Defines a markdown math block, enclosed in a pair of dollar symbols.
--------------------------	--

euporie.markdown.blocks.math.MathBlock

class euporie.markdown.blocks.math.MathBlock
Defines a markdown math block, enclosed in a pair of dollar symbols.

class euporie.markdown.blocks.math.MathBlock
Bases: commonmark.blocks.Block

Defines a markdown math block, enclosed in a pair of dollar symbols.

accepts_lines = True

static **can_contain**(*t: str*) → bool
Ensures the math block does not contain other nodes.

static **continue_**(*parser: Parser, container: Node*) → int
Checks for the end of the math block.

static **finalize**(*parser: Parser, block: Node*) → None
Ends the math block.

euporie.markdown.blocks.tables

An extension for python commonmark to include markdown tables.

Based on <https://github.com/GovReady/CommonMark-py-Extensions>

Classes

<code>Table()</code>	Define a new markdown Table block.
----------------------	------------------------------------

euporie.markdown.blocks.tables.Table

class euporie.markdown.blocks.tables.Table

Define a new markdown Table block.

Define a new Table class that handles incoming table lines, modeled a bit after the Blockquote, which allows continuation lines so long as they start with the symbol. Also has `accepts_lines` to suck in everything within it as raw data. Accept : as a continuation symbol for Github-flavored Markdown table column alignment.

class euporie.markdown.blocks.tables.Table

Bases: `commonmark.blocks.Block`

Define a new markdown Table block.

Define a new Table class that handles incoming table lines, modeled a bit after the Blockquote, which allows continuation lines so long as they start with the symbol. Also has `accepts_lines` to suck in everything within it as raw data. Accept : as a continuation symbol for Github-flavored Markdown table column alignment.

accepts_lines = True

static `can_contain(t)`

static `continue_(parser: Parser, container: Node) → int`

Checks for the end of the table block.

static `finalize(parser: Parser, block: Node) → None`

Split the table content into rows and columns, with each line a new row.

Classes

<code>BlockStarts()</code>	Subclasses the BlockStarts class, implementing extension methods.
----------------------------	---

euporie.markdown.blocks.BlockStarts

class euporie.markdown.blocks.BlockStarts

Subclasses the BlockStarts class, implementing extension methods.

class euporie.markdown.blocks.BlockStarts

Bases: `commonmark.blocks.BlockStarts`

Subclasses the BlockStarts class, implementing extension methods.

METHODS: List[str] = ['block_quote', 'atx_heading', 'fenced_code_block', 'html_block', 'setext_heading', 'thematic_break', 'list_item', 'indented_code_block']

static `atx_heading(parser, container=None)`

static `block_quote(parser, container=None)`

static `fenced_code_block(parser, container=None)`

```

static html_block(parser, container=None)
static indented_code_block(parser, container=None)
static list_item(parser, container=None)
static math_block(parser: Any, container: Optional[Any] = None) → int
    Defines the start of a block of maths.
static setext_heading(parser, container=None)
static table(parser: Any, container: Optional[Any]) → int
    Defines the start of a table.
static thematic_break(parser, container=None)

class euporie.markdown.blocks.MathBlock
    Bases: commonmark.blocks.Block

    Defines a markdown math block, enclosed in a pair of dollar symbols.

    accepts_lines = True

    static can_contain(t: str) → bool
        Ensures the math block does not contain other nodes.

    static continue_(parser: Parser, container: Node) → int
        Checks for the end of the math block.

    static finalize(parser: Parser, block: Node) → None
        Ends the math block.

class euporie.markdown.blocks.Table
    Bases: commonmark.blocks.Block

    Define a new markdown Table block.

    Define a new Table class that handles incoming table lines, modeled a bit after the Blockquote, which allows
    continuation lines so long as they start with the symbol. Also has accepts_lines to suck in everything within it
    as raw data. Accept : as a continuation symbol for Github-flavored Markdown table column alignment.

    accepts_lines = True

    static can_contain(t)

    static continue_(parser: Parser, container: Node) → int
        Checks for the end of the table block.

    static finalize(parser: Parser, block: Node) → None
        Split the table content into rows and columns, with each line a new row.

```

euporie.markdown.inlines

Defines additional inline nodes for the python commonmark parser.

Classes

<code>InlineParser</code> ([options])	An extension of the commonmark inline parser.
---------------------------------------	---

`euporie.markdown.inlines.InlineParser`

class `euporie.markdown.inlines.InlineParser`(*options*={})

An extension of the commonmark inline parser.

Includes support for: - Inline maths delimited with dollar signs (\$)

class `euporie.markdown.inlines.InlineParser`(*options*={})

Bases: `commonmark.inlines.InlineParser`

An extension of the commonmark inline parser.

Includes support for: - Inline maths delimited with dollar signs (\$)

addBracket(*node*, *index*, *image*)

handleDelim(*cc*, *block*)

Handle a delimiter marker for emphasis or a quote.

match(*regexString*)

If *regexString* matches at current position in the subject, advance position in subject and return the match; otherwise return None.

parse(*block*)

Parse string content in block into inline children, using *refmap* to resolve references.

parseAutolink(*block*)

Attempt to parse an autolink (URL or email in pointy brackets).

parseBackslash(*block*)

Parse a backslash-escaped special character, adding either the escaped character, a hard line break (if the backslash is followed by a newline), or a literal backslash to the block's children. Assumes current character is a backslash.

parseBackticks(*block*)

Attempt to parse backticks, adding either a backtick code span or a literal sequence of backticks to the 'inlines' list.

parseBang(*block*)

If next character is [, and ! delimiter to delimiter stack and add a text node to block's children. Otherwise just add a text node.

parseCloseBracket(*block*)

Try to match close bracket against an opening in the delimiter stack. Add either a link or image, or a plain [character, to block's children. If there is a matching delimiter, remove it from the delimiter stack.

parseEntity(*block*)

Attempt to parse an entity.

parseHtmlTag(*block*)

Attempt to parse a raw HTML tag.

parseInline(*block*: Any) → bool

Override the inline parsing function to include parsing inline math.

parseInlines(*block*)

Parse string content in block into inline children, using refmap to resolve references.

parseLinkDestination()

Attempt to parse link destination, returning the string or None if no match.

parseLinkLabel()

Attempt to parse a link label, returning number of characters parsed.

parseLinkTitle()

Attempt to parse link title (sans quotes), returning the string or None if no match.

parseMath(*block: Any*) → bool

Attempt to parse inline math code between pairs of '\$'.

parseNewline(*block*)

Parse a newline. If it was preceded by two spaces, return a hard line break; otherwise a soft line break.

parseOpenBracket(*block*)

Add open bracket to delimiter stack and add a text node to block's children.

parseReference(*s, refmap*)

Attempt to parse a link reference, modifying refmap.

parseString(*block*)

Parse a run of ordinary characters, or a single character with a special meaning in markdown, as a plain string.

peek()

Returns the character at the current subject position, or None if there are no more characters.

pos: int

processEmphasis(*stack_bottom*)

removeBracket()

removeDelimiter(*delim*)

static removeDelimitersBetween(*bottom, top*)

scanDelims(*c*)

Scan a sequence of characters == *c*, and return information about the number of delimiters and whether they are positioned such that they can open and/or close emphasis or strong emphasis. A utility function for strong/emph parsing.

spnl()

Parse zero or more space characters, including at most one newline.

euporie.markdown.parser

Adds extensions to the python commonmark parser.

Classes

<code>Parser([options])</code>	Subclasses the commonmark parser to include tables and math blocks.
--------------------------------	---

`euporie.markdown.parser.Parser`

class `euporie.markdown.parser.Parser`(*options: Optional[dict] = None*)

Subclasses the commonmark parser to include tables and math blocks.

class `euporie.markdown.parser.Parser`(*options: Optional[dict] = None*)

Bases: `commonmark.blocks.Parser`

Subclasses the commonmark parser to include tables and math blocks.

add_child(*tag, offset*)

Add block of type *tag* as a child of the tip. If the tip can't accept children, close and finalize it and try its parent, and so on til we find a block that can accept children.

add_line()

Add a line to the block at the tip. We assume the tip can accept lines – that check should be done before calling this.

advance_next_nonspace()

advance_offset(*count, columns*)

```
blocks = {'block_quote': <class 'commonmark.blocks.BlockQuote'>, 'code_block':  
<class 'commonmark.blocks.CodeBlock'>, 'document': <class  
'commonmark.blocks.Document'>, 'heading': <class 'commonmark.blocks.Heading'>,  
'html_block': <class 'commonmark.blocks.HtmlBlock'>, 'item': <class  
'commonmark.blocks.Item'>, 'list': <class 'commonmark.blocks.List'>, 'paragraph':  
<class 'commonmark.blocks.Paragraph'>, 'thematic_break': <class  
'commonmark.blocks.ThematicBreak'>}
```

close_unmatched_blocks()

Finalize and close any unmatched blocks.

finalize(*block, line_number*)

Finalize a block. Close it and do any necessary postprocessing, e.g. creating `string_content` from strings, setting the 'tight' or 'loose' status of a list, and parsing the beginnings of paragraphs for reference definitions. Reset the tip to the parent of the closed block.

find_next_nonspace()

incorporate_line(*ln*)

Analyze a line of text and update the document appropriately.

We parse markdown text by calling this on each line of input, then finalizing the document.

parse(*my_input*)

The main parsing function. Returns a parsed document AST.

process_inlines(*block*)

Walk through a block & children recursively, parsing string content into inline content where appropriate.

euporie.markdown.rich

Adds extends :py:class`rich.markdown.Markdown` with tables and LaTeX.

Classes

<code>LatexBlock(justify)</code>	Render a block of LaTeX centered as a paragraph.
<code>LatexElement()</code>	A latex element which renders text as LaTeX.
<code>LatexInline()</code>	Render a LaTeX in a paragraph.
<code>Markdown(markup[, code_theme, justify, ...])</code>	Rich's markdown with additional elements.
<code>Table(table_node)</code>	A table.

euporie.markdown.rich.LatexBlock

```
class euporie.markdown.rich.LatexBlock(justify: Literal['default', 'left', 'center', 'right', 'full'])
    Render a block of LaTeX centered as a paragraph.
```

euporie.markdown.rich.LatexElement

```
class euporie.markdown.rich.LatexElement
    A latex element which renders text as LaTeX.
```

euporie.markdown.rich.LatexInline

```
class euporie.markdown.rich.LatexInline
    Render a LaTeX in a paragraph.
```

euporie.markdown.rich.Markdown

```
class euporie.markdown.rich.Markdown(markup: str, code_theme: str = 'monokai', justify:
    Optional[Literal['default', 'left', 'center', 'right', 'full']] = None, style:
    Union[str, rich.style.Style] = 'none', hyperlinks: bool = True,
    inline_code_lexer: Optional[str] = None, inline_code_theme:
    Optional[str] = None)
    Rich's markdown with additional elements.
```

euporie.markdown.rich.Table

```
class euporie.markdown.rich.Table(table_node: TableNode)
    A table.

class euporie.markdown.rich.LatexBlock(justify: Literal['default', 'left', 'center', 'right', 'full'])
    Bases: rich.markdown.Paragraph, euporie.markdown.rich.LatexElement

    Render a block of LaTeX centered as a paragraph.

    classmethod create(markdown: rich.markdown.Markdown, node: rich.markdown.MarkdownElement) →
        rich.markdown.Paragraph
        Factory to create markdown element,
```

Parameters

- **markdown** (*Markdown*) – The parent Markdown object.
- **node** (*Any*) – A node from Pygments.

Returns A new markdown element

Return type *MarkdownElement*

justify: *JustifyMethod*

new_line: *ClassVar[bool]* = *True*

on_child_close(*context: rich.markdown.MarkdownContext, child: rich.markdown.MarkdownElement*) → *bool*

Called when a child element is closed.

This method allows a parent element to take over rendering of its children.

Parameters

- **context** (*MarkdownContext*) – The markdown context.
- **child** (*MarkdownElement*) – The child markdown element.

Returns Return True to render the element, or False to not render the element.

Return type *bool*

on_enter(*context: rich.markdown.MarkdownContext*) → *None*

Called when the node is entered.

Parameters **context** (*MarkdownContext*) – The markdown context.

on_leave(*context: rich.markdown.MarkdownContext*) → *None*

Called when the parser leaves the element.

Parameters **context** (*MarkdownContext*) – [description]

on_text(*context: MarkdownContext, text: TextType*) → *None*

Converts LaTeX source text to rendered LaTeX text.

style_name = *'markdown.paragraph'*

class *euporie.markdown.rich.LatexElement*

Bases: *rich.markdown.TextElement*

A latex element which renders text as LaTeX.

classmethod **create**(*markdown: rich.markdown.Markdown, node: Any*) → *rich.markdown.MarkdownElement*

Factory to create markdown element,

Parameters

- **markdown** (*Markdown*) – The parent Markdown object.
- **node** (*Any*) – A node from Pygments.

Returns A new markdown element

Return type *MarkdownElement*

new_line: *ClassVar[bool]* = *True*

on_child_close(*context*: *rich.markdown.MarkdownContext*, *child*: *rich.markdown.MarkdownElement*) → *bool*

Called when a child element is closed.

This method allows a parent element to take over rendering of its children.

Parameters

- **context** (*MarkdownContext*) – The markdown context.
- **child** (*MarkdownElement*) – The child markdown element.

Returns Return True to render the element, or False to not render the element.

Return type *bool*

on_enter(*context*: *rich.markdown.MarkdownContext*) → *None*

Called when the node is entered.

Parameters **context** (*MarkdownContext*) – The markdown context.

on_leave(*context*: *rich.markdown.MarkdownContext*) → *None*

Called when the parser leaves the element.

Parameters **context** (*MarkdownContext*) – [description]

on_text(*context*: *MarkdownContext*, *text*: *TextType*) → *None*

Converts LaTeX source text to rendered LaTeX text.

style_name = 'none'

class *euporie.markdown.rich.LatexInline*

Bases: *euporie.markdown.rich.LatexElement*

Render a LaTeX in a paragraph.

classmethod **create**(*markdown*: *rich.markdown.Markdown*, *node*: *Any*) → *rich.markdown.MarkdownElement*

Factory to create markdown element,

Parameters

- **markdown** (*Markdown*) – The parent Markdown object.
- **node** (*Any*) – A node from Pygments.

Returns A new markdown element

Return type *MarkdownElement*

new_line: *ClassVar*[*bool*] = *False*

on_child_close(*context*: *rich.markdown.MarkdownContext*, *child*: *rich.markdown.MarkdownElement*) → *bool*

Called when a child element is closed.

This method allows a parent element to take over rendering of its children.

Parameters

- **context** (*MarkdownContext*) – The markdown context.
- **child** (*MarkdownElement*) – The child markdown element.

Returns Return True to render the element, or False to not render the element.

Return type *bool*

on_enter(context: *rich.markdown.MarkdownContext*) → *None*

Called when the node is entered.

Parameters context (*MarkdownContext*) – The markdown context.

on_leave(context: *rich.markdown.MarkdownContext*) → *None*

Called when the parser leaves the element.

Parameters context (*MarkdownContext*) – [description]

on_text(context: *MarkdownContext*, text: *TextType*) → *None*

Converts LaTeX source text to rendered LaTeX text.

style_name = 'none'

```
class euporie.markdown.rich.Markdown(markup: str, code_theme: str = 'monokai', justify:
    Optional[Literal['default', 'left', 'center', 'right', 'full']] = None, style:
    Union[str, rich.style.Style] = 'none', hyperlinks: bool = True,
    inline_code_lexer: Optional[str] = None, inline_code_theme:
    Optional[str] = None)
```

Bases: *rich.markdown.Markdown*

Rich's markdown with additional elements.

```
elements: ClassVar[Dict[str, Type[MarkdownElement]]] = {'block_quote': <class
'rich.markdown.BlockQuote'>, 'code_block': <class 'rich.markdown.CodeBlock'>,
'heading': <class 'rich.markdown.Heading'>, 'image': <class
'rich.markdown.ImageItem'>, 'item': <class 'rich.markdown.ListItem'>, 'list':
<class 'rich.markdown.ListElement'>, 'math': <class
'euporie.markdown.rich.LatexInline'>, 'math_block': <class
'euporie.markdown.rich.LatexBlock'>, 'paragraph': <class
'rich.markdown.Paragraph'>, 'table': <class 'euporie.markdown.rich.Table'>,
'thematic_break': <class 'rich.markdown.HorizontalRule'>}
```

```
inlines = {'code', 'emph', 'strike', 'strong'}
```

```
justify: Optional[JustifyMethod]
```

```
class euporie.markdown.rich.Table(table_node: TableNode)
```

Bases: *rich.markdown.MarkdownElement*

A table.

```
classmethod create(markdown: RichMarkdown, table_node: TableNode) → Table
```

Instantiates and returns a rich markdown table.

```
dumb_console = <console width=80 None>
```

```
get_markdown_width(md: euporie.markdown.rich.Markdown) → int
```

Get the width of markdown text.

```
new_line: ClassVar[bool] = True
```

```
static node_to_md(node: Node, justify: JustifyMethod = None) → Markdown
```

Create a rich markdown object, then add the parsed node to it.

```
on_child_close(context: rich.markdown.MarkdownContext, child: rich.markdown.MarkdownElement) →
    bool
```

Called when a child element is closed.

This method allows a parent element to take over rendering of its children.

Parameters

- **context** (*MarkdownContext*) – The markdown context.
- **child** (*MarkdownElement*) – The child markdown element.

Returns Return True to render the element, or False to not render the element.

Return type `bool`

on_enter(*context: rich.markdown.MarkdownContext*) → *None*

Called when the node is entered.

Parameters **context** (*MarkdownContext*) – The markdown context.

on_leave(*context: rich.markdown.MarkdownContext*) → *None*

Called when the parser leaves the element.

Parameters **context** (*MarkdownContext*) – [description]

on_text(*context: rich.markdown.MarkdownContext, text: Union[str, rich.text.Text]*) → *None*

Called when text is parsed.

Parameters **context** (*MarkdownContext*) – The markdown context.

style_name = `'markdown.table'`

class `euporie.markdown.InlineParser(options={})`

Bases: `commonmark.inlines.InlineParser`

An extension of the commonmark inline parser.

Includes support for: - Inline maths delimited with dollar signs (\$)

addBracket(*node, index, image*)

handleDelim(*cc, block*)

Handle a delimiter marker for emphasis or a quote.

match(*regexString*)

If regexString matches at current position in the subject, advance position in subject and return the match; otherwise return None.

parse(*block*)

Parse string content in block into inline children, using refmap to resolve references.

parseAutolink(*block*)

Attempt to parse an autolink (URL or email in pointy brackets).

parseBackslash(*block*)

Parse a backslash-escaped special character, adding either the escaped character, a hard line break (if the backslash is followed by a newline), or a literal backslash to the block's children. Assumes current character is a backslash.

parseBackticks(*block*)

Attempt to parse backticks, adding either a backtick code span or a literal sequence of backticks to the 'inlines' list.

parseBang(*block*)

If next character is [, and ! delimiter to delimiter stack and add a text node to block's children. Otherwise just add a text node.

parseCloseBracket(*block*)

Try to match close bracket against an opening in the delimiter stack. Add either a link or image, or a plain [character, to block's children. If there is a matching delimiter, remove it from the delimiter stack.

parseEntity(*block*)
Attempt to parse an entity.

parseHtmlTag(*block*)
Attempt to parse a raw HTML tag.

parseInline(*block: Any*) → *bool*
Override the inline parsing function to include parsing inline math.

parseInlines(*block*)
Parse string content in block into inline children, using refmap to resolve references.

parseLinkDestination()
Attempt to parse link destination, returning the string or None if no match.

parseLinkLabel()
Attempt to parse a link label, returning number of characters parsed.

parseLinkTitle()
Attempt to parse link title (sans quotes), returning the string or None if no match.

parseMath(*block: Any*) → *bool*
Attempt to parse inline math code between pairs of '\$'.

parseNewline(*block*)
Parse a newline. If it was preceded by two spaces, return a hard line break; otherwise a soft line break.

parseOpenBracket(*block*)
Add open bracket to delimiter stack and add a text node to block's children.

parseReference(*s, refmap*)
Attempt to parse a link reference, modifying refmap.

parseString(*block*)
Parse a run of ordinary characters, or a single character with a special meaning in markdown, as a plain string.

peek()
Returns the character at the current subject position, or None if there are no more characters.

pos: *int*

processEmphasis(*stack_bottom*)

removeBracket()

removeDelimiter(*delim*)

static removeDelimitersBetween(*bottom, top*)

scanDelims(*c*)
Scan a sequence of characters == *c*, and return information about the number of delimiters and whether they are positioned such that they can open and/or close emphasis or strong emphasis. A utility function for strong/emph parsing.

spnl()
Parse zero or more space characters, including at most one newline.

class euporie.markdown.**LatexBlock**(*justify: Literal['default', 'left', 'center', 'right', 'full']*)
Bases: *rich.markdown.Paragraph*, *euporie.markdown.rich.LatexElement*
Render a block of LaTeX centered as a paragraph.

classmethod create(*markdown: rich.markdown.Markdown, node: rich.markdown.MarkdownElement*) → *rich.markdown.Paragraph*

Factory to create markdown element,

Parameters

- **markdown** (*Markdown*) – The parent Markdown object.
- **node** (*Any*) – A node from Pygments.

Returns A new markdown element

Return type *MarkdownElement*

justify: *JustifyMethod*

new_line: *ClassVar[bool]* = *True*

on_child_close(*context: rich.markdown.MarkdownContext, child: rich.markdown.MarkdownElement*) → *bool*

Called when a child element is closed.

This method allows a parent element to take over rendering of its children.

Parameters

- **context** (*MarkdownContext*) – The markdown context.
- **child** (*MarkdownElement*) – The child markdown element.

Returns Return True to render the element, or False to not render the element.

Return type *bool*

on_enter(*context: rich.markdown.MarkdownContext*) → *None*

Called when the node is entered.

Parameters **context** (*MarkdownContext*) – The markdown context.

on_leave(*context: rich.markdown.MarkdownContext*) → *None*

Called when the parser leaves the element.

Parameters **context** (*MarkdownContext*) – [description]

on_text(*context: MarkdownContext, text: TextType*) → *None*

Converts LaTeX source text to rendered LaTeX text.

style_name = *'markdown.paragraph'*

class *euporie.markdown.LatexElement*

Bases: *rich.markdown.TextElement*

A latex element which renders text as LaTeX.

classmethod create(*markdown: rich.markdown.Markdown, node: Any*) → *rich.markdown.MarkdownElement*

Factory to create markdown element,

Parameters

- **markdown** (*Markdown*) – The parent Markdown object.
- **node** (*Any*) – A node from Pygments.

Returns A new markdown element

Return type *MarkdownElement*

new_line: ClassVar[bool] = True

on_child_close(context: *rich.markdown.MarkdownContext*, child: *rich.markdown.MarkdownElement*) → bool

Called when a child element is closed.

This method allows a parent element to take over rendering of its children.

Parameters

- **context** (*MarkdownContext*) – The markdown context.
- **child** (*MarkdownElement*) – The child markdown element.

Returns Return True to render the element, or False to not render the element.

Return type bool

on_enter(context: *rich.markdown.MarkdownContext*) → None

Called when the node is entered.

Parameters **context** (*MarkdownContext*) – The markdown context.

on_leave(context: *rich.markdown.MarkdownContext*) → None

Called when the parser leaves the element.

Parameters **context** (*MarkdownContext*) – [description]

on_text(context: *MarkdownContext*, text: *TextType*) → None

Converts LaTeX source text to rendered LaTeX text.

style_name = 'none'

class euporie.markdown.LatexInline

Bases: *euporie.markdown.rich.LatexElement*

Render a LaTeX in a paragraph.

classmethod **create**(markdown: *rich.markdown.Markdown*, node: *Any*) → *rich.markdown.MarkdownElement*

Factory to create markdown element,

Parameters

- **markdown** (*Markdown*) – The parent Markdown object.
- **node** (*Any*) – A node from Pygments.

Returns A new markdown element

Return type *MarkdownElement*

new_line: ClassVar[bool] = False

on_child_close(context: *rich.markdown.MarkdownContext*, child: *rich.markdown.MarkdownElement*) → bool

Called when a child element is closed.

This method allows a parent element to take over rendering of its children.

Parameters

- **context** (*MarkdownContext*) – The markdown context.
- **child** (*MarkdownElement*) – The child markdown element.

Returns Return True to render the element, or False to not render the element.

Return type `bool`

on_enter(*context: rich.markdown.MarkdownContext*) → `None`

Called when the node is entered.

Parameters **context** (*MarkdownContext*) – The markdown context.

on_leave(*context: rich.markdown.MarkdownContext*) → `None`

Called when the parser leaves the element.

Parameters **context** (*MarkdownContext*) – [description]

on_text(*context: MarkdownContext, text: TextType*) → `None`

Converts LaTeX source text to rendered LaTeX text.

style_name = `'none'`

```
class euporie.markdown.Markdown(markup: str, code_theme: str = 'monokai', justify:
    Optional[Literal['default', 'left', 'center', 'right', 'full']] = None, style:
    Union[str, rich.style.Style] = 'none', hyperlinks: bool = True,
    inline_code_lexer: Optional[str] = None, inline_code_theme:
    Optional[str] = None)
```

Bases: `rich.markdown.Markdown`

Rich's markdown with additional elements.

```
elements: ClassVar[Dict[str, Type[MarkdownElement]]] = {'block_quote': <class
'rich.markdown.BlockQuote'>, 'code_block': <class 'rich.markdown.CodeBlock'>,
'heading': <class 'rich.markdown.Heading'>, 'image': <class
'rich.markdown.ImageItem'>, 'item': <class 'rich.markdown.ListItem'>, 'list':
<class 'rich.markdown.ListElement'>, 'math': <class
'euporie.markdown.rich.LatexInline'>, 'math_block': <class
'euporie.markdown.rich.LatexBlock'>, 'paragraph': <class
'rich.markdown.Paragraph'>, 'table': <class 'euporie.markdown.rich.Table'>,
'thematic_break': <class 'rich.markdown.HorizontalRule'>}
```

```
inlines = {'code', 'emph', 'strike', 'strong'}
```

```
class euporie.markdown.Parser(options: Optional[dict] = None)
```

Bases: `commonmark.blocks.Parser`

Subclasses the commonmark parser to include tables and math blocks.

add_child(*tag, offset*)

Add block of type tag as a child of the tip. If the tip can't accept children, close and finalize it and try its parent, and so on til we find a block that can accept children.

add_line()

Add a line to the block at the tip. We assume the tip can accept lines – that check should be done before calling this.

advance_next_nonspace()

advance_offset(*count, columns*)

```
blocks = {'block_quote': <class 'commonmark.blocks.BlockQuote'>, 'code_block':
<class 'commonmark.blocks.CodeBlock'>, 'document': <class
'commonmark.blocks.Document'>, 'heading': <class 'commonmark.blocks.Heading'>,
'html_block': <class 'commonmark.blocks.HtmlBlock'>, 'item': <class
'commonmark.blocks.Item'>, 'list': <class 'commonmark.blocks.List'>, 'paragraph':
<class 'commonmark.blocks.Paragraph'>, 'thematic_break': <class
'commonmark.blocks.ThematicBreak'>}
```

close_unmatched_blocks()

Finalize and close any unmatched blocks.

finalize(*block*, *line_number*)

Finalize a block. Close it and do any necessary postprocessing, e.g. creating `string_content` from strings, setting the ‘tight’ or ‘loose’ status of a list, and parsing the beginnings of paragraphs for reference definitions. Reset the tip to the parent of the closed block.

find_next_nonspace()

incorporate_line(*ln*)

Analyze a line of text and update the document appropriately.

We parse markdown text by calling this on each line of input, then finalizing the document.

parse(*my_input*)

The main parsing function. Returns a parsed document AST.

process_inlines(*block*)

Walk through a block & children recursively, parsing string content into inline content where appropriate.

class euporie.markdown.**Table**(*table_node*: *TableNode*)

Bases: rich.markdown.MarkdownElement

A table.

classmethod **create**(*markdown*: *RichMarkdown*, *table_node*: *TableNode*) → *Table*

Instantiates and returns a rich markdown table.

dumb_console = <console width=80 None>

get_markdown_width(*md*: euporie.markdown.rich.Markdown) → int

Get the width of markdown text.

new_line: ClassVar[bool] = True

static **node_to_md**(*node*: *Node*, *justify*: *JustifyMethod* = None) → *Markdown*

Create a rich markdown object, then add the parsed node to it.

on_child_close(*context*: rich.markdown.MarkdownContext, *child*: rich.markdown.MarkdownElement) → bool

Called when a child element is closed.

This method allows a parent element to take over rendering of its children.

Parameters

- **context** (*MarkdownContext*) – The markdown context.
- **child** (*MarkdownElement*) – The child markdown element.

Returns Return True to render the element, or False to not render the element.

Return type bool

on_enter(*context*: rich.markdown.MarkdownContext) → None

Called when the node is entered.

Parameters **context** (*MarkdownContext*) – The markdown context.

on_leave(*context*: rich.markdown.MarkdownContext) → None

Called when the parser leaves the element.

Parameters **context** (*MarkdownContext*) – [description]

on_text(*context: rich.markdown.MarkdownContext, text: Union[str, rich.text.Text]*) → None
 Called when text is parsed.

Parameters **context** (*MarkdownContext*) – The markdown context.

style_name = 'markdown.table'

5.8.17 euporie.menu

Defines the application's menu bar and them menu's contents.

Modules

<i>euporie.menu.bar</i>	Defines the application menu.
<i>euporie.menu.contents</i>	Defines the application's menu structure.
<i>euporie.menu.item</i>	Defines the application menu.

euporie.menu.bar

Defines the application menu.

Classes

<i>MenuContainer</i> (body, menu_items[, floats, ...])	A container to hold the menubar and main application body.
--	--

euporie.menu.bar.MenuContainer

```
class euporie.menu.bar.MenuContainer(body: AnyContainer, menu_items: list[PtkMenuItem], floats:
    Optional[list[Float]] = None, key_bindings:
    Optional[KeyBindingsBase] = None, left:
    Optional[Sequence[AnyContainer]] = None, right:
    Optional[Sequence[AnyContainer]] = None)
```

A container to hold the menubar and main application body.

```
class euporie.menu.bar.MenuContainer(body: AnyContainer, menu_items: list[PtkMenuItem], floats:
    Optional[list[Float]] = None, key_bindings:
    Optional[KeyBindingsBase] = None, left:
    Optional[Sequence[AnyContainer]] = None, right:
    Optional[Sequence[AnyContainer]] = None)
```

Bases: `prompt_toolkit.widgets.menus.MenuContainer`

A container to hold the menubar and main application body.

property floats: Optional[List[prompt_toolkit.layout.containers.Float]]

statusbar_fields() → tuple[list[AnyFormattedText], list[AnyFormattedText]]

Return the description of the currently selected menu item.

euporie.menu.contents

Defines the application's menu structure.

Functions

<code>load_menu_items()</code>	Loads the list of menu items to display in the menu.
--------------------------------	--

euporie.menu.contents.load_menu_items

`euporie.menu.contents.load_menu_items()` → `list[euporie.menu.item.MenuItem]`
Loads the list of menu items to display in the menu.

`euporie.menu.contents.load_menu_items()` → `list[euporie.menu.item.MenuItem]`
Loads the list of menu items to display in the menu.

euporie.menu.item

Defines the application menu.

Classes

<code>MenuItem([formatted_text, description, ...])</code>	A prompt-toolkit compatible menu item with more advanced capabilities.
---	--

euporie.menu.item.MenuItem

class `euporie.menu.item.MenuItem`(*formatted_text*: `AnyFormattedText` = "", *description*: `str` = "", *separator*: `bool` = `False`, *handler*: `Optional[Callable[[], None]]` = `None`, *children*: `Optional[list[PtkMenuItem]]` = `None`, *shortcut*: `Optional[Sequence[Union[Keys, str]]]` = `None`, *hidden*: `FilterOrBool` = `False`, *disabled*: `FilterOrBool` = `False`, *toggled*: `Optional[Filter]` = `None`)

A prompt-toolkit compatible menu item with more advanced capabilities.

It can use a function to generate formatted text to display, display a checkmark if a condition is true, and disable the handler if a condition is met.

class `euporie.menu.item.MenuItem`(*formatted_text*: `AnyFormattedText` = "", *description*: `str` = "", *separator*: `bool` = `False`, *handler*: `Optional[Callable[[], None]]` = `None`, *children*: `Optional[list[PtkMenuItem]]` = `None`, *shortcut*: `Optional[Sequence[Union[Keys, str]]]` = `None`, *hidden*: `FilterOrBool` = `False`, *disabled*: `FilterOrBool` = `False`, *toggled*: `Optional[Filter]` = `None`)

Bases: `prompt_toolkit.widgets.menus.MenuItem`

A prompt-toolkit compatible menu item with more advanced capabilities.

It can use a function to generate formatted text to display, display a checkmark if a condition is true, and disable the handler if a condition is met.

property disabled: `bool`

Determine if the menu item is disabled.

```

property formatted_text: StyleAndTextTuples
    Generate the formatted text for this menu item.

classmethod from_command(command: Command) → MenuItem
    Create a menu item from a command.

property has_toggles: bool
    Returns true if any child items have a toggle state.

property prefix: StyleAndTextTuples
    The item's prefix.

    Formatted text that will be displayed before the item's main text. All prefixes in a menu are left aligned and
    padded to take up equal width.

    Returns Formatted text

property prefix_width: int
    The maximum width of the item's children's prefixes.

property shortcut_str: str
    A string representing the item's main keyboard shortcut.

property suffix: StyleAndTextTuples
    The item's suffix.

    Formatted text that will be displayed aligned right after them item's main text.

    Returns Formatted text

property suffix_width: int
    The maximum width of the item's children's suffixes.

property text: str
    Return plain text version of the item's formatted text.

property width: int
    The maximum width of the item's children.

class euporie.menu.MenuContainer(body: AnyContainer, menu_items: list[PtkMenuItem], floats:
    Optional[list[Float]] = None, key_bindings: Optional[KeyBindingsBase]
    = None, left: Optional[Sequence[AnyContainer]] = None, right:
    Optional[Sequence[AnyContainer]] = None)

    Bases: prompt_toolkit.widgets.menus.MenuContainer

    A container to hold the menubar and main application body.

property floats: Optional[List[prompt_toolkit.layout.containers.Float]]

statusbar_fields() → tuple[list[AnyFormattedText], list[AnyFormattedText]]
    Return the description of the currently selected menu item.

class euporie.menu.MenuItem(formatted_text: AnyFormattedText = "", description: str = "", separator: bool =
    False, handler: Optional[Callable[[], None]] = None, children:
    Optional[list[PtkMenuItem]] = None, shortcut:
    Optional[Sequence[Union[Keys, str]]] = None, hidden: FilterOrBool = False,
    disabled: FilterOrBool = False, toggled: Optional[Filter] = None)

    Bases: prompt_toolkit.widgets.menus.MenuItem

    A prompt-toolkit compatible menu item with more advanced capabilities.

    It can use a function to generate formatted text to display, display a checkmark if a condition is true, and disable
    the handler if a condition is met.

```

property disabled: `bool`

Determine if the menu item is disabled.

property formatted_text: `StyleAndTextTuples`

Generate the formatted text for this menu item.

classmethod from_command(*command*: `Command`) → *MenuItem*

Create a menu item from a command.

property has_toggles: `bool`

Returns true if any child items have a toggle state.

property prefix: `StyleAndTextTuples`

The item's prefix.

Formatted text that will be displayed before the item's main text. All prefixes in a menu are left aligned and padded to take up equal width.

Returns Formatted text

property prefix_width: `int`

The maximum width of the item's children's prefixes.

property shortcut_str: `str`

A string representing the item's main keyboard shortcut.

property suffix: `StyleAndTextTuples`

The item's suffix.

Formatted text that will be displayed aligned right after the item's main text.

Returns Formatted text

property suffix_width: `int`

The maximum width of the item's children's suffixes.

property text: `str`

Return plain text version of the item's formatted text.

property width: `int`

The maximum width of the item's children.

5.8.18 euporie.notebook

Contains the main class for a notebook file.

Classes

DumpKernelNotebook(path[, app])

DumpNotebook(path[, app])

KernelNotebook(path[, app])

Notebook(path[, app])

The main notebook container class.

TuiNotebook(path[, app])

euporie.notebook.DumpKernelNotebook

```
class euporie.notebook.DumpKernelNotebook(path: Path, app: Optional[EuporieApp] = None)
```

euporie.notebook.DumpNotebook

```
class euporie.notebook.DumpNotebook(path: Path, app: Optional[EuporieApp] = None)
```

euporie.notebook.KernelNotebook

```
class euporie.notebook.KernelNotebook(path: Path, app: Optional[EuporieApp] = None)
```

euporie.notebook.Notebook

```
class euporie.notebook.Notebook(path: Path, app: Optional[EuporieApp] = None)
```

The main notebook container class.

euporie.notebook.TuiNotebook

```
class euporie.notebook.TuiNotebook(path: Path, app: Optional[EuporieApp] = None)
```

```
class euporie.notebook.Notebook(path: Path, app: Optional[EuporieApp] = None)
```

Bases: [euporie.tab.Tab](#)

The main notebook container class.

add(index: int) → None
Creates a new cell at a given index.

Parameters **index** – The position at which to insert a new cell

close(cb: Optional[Callable]) → None
Function to close a tab with a callback.

Parameters **cb** – A function to call after the tab is closed.

container: _Split

copy(slice_: slice) → None
Add a copy of this cell to the *Notebook*'s clipboard.

cut(slice_: slice) → None
Remove a cell from the notebook and add it to the *Notebook*'s clipboard.

delete(slice_: Optional[slice] = None) → None
Delete a cell from the notebook.

focus() → None
Focuses the tab.

get_cell_by_id(cell_id: str) → Optional[Cell]
Returns a reference to the *Cell* container with a given cell id.

hide_pager() → None
Closes the pager.

lang_file_ext() → str
Return the file extension for scripts in the notebook's language.

merge(*slice_*: *Optional[slice] = None*) → *None*

Merge two or more cells.

move(*n*: *int*, *slice_*: *Optional[slice] = None*) → *None*

Move a slice of cells up or down.

Parameters

- **slice** – A slice describing the cell indices to move
- **n** – The amount to move them by

paste(*index*: *int*) → *None*

Append the contents of the *Notebook*'s clipboard below the first selected cell.

refresh(*slice_*: *Optional[slice] = None*, *scroll*: *bool = True*) → *None*

Refresh the notebook display.

rendered_cells() → *list[euporie.cell.Cell]*

Return a list of *Cell* generator functions for the notebooks' cells.

run_cell(*cell*: *euporie.cell.Cell*, *wait*: *bool = False*) → *None*

Runs a cell in the notebook.

save() → *None*

Write the notebook's JSON to the current notebook's file.

split_cell(*cell*: *euporie.cell.Cell*, *cursor_position*: *int*) → *None*

Splits a cell into two at the given cursor position.

Parameters

- **cell** – The rendered cell to split
- **cursor_position** – The position at which to split the cell

statusbar_fields() → *tuple[Sequence[AnyFormattedText], Sequence[AnyFormattedText]]*

Returns a list of statusbar field values shown then this tab is active.

property title: *str*

Return the tab title.

5.8.19 euporie.output

Defines an output container and the controls used within in.

Modules

<i>euporie.output.container</i>	Container class for cell output.
<i>euporie.output.control</i>	Defines custom controls which re-render on resize.

euporie.output.container

Container class for cell output.

Functions

<code>get_dims(data, format_[, px, py, fg, bg])</code>	Get the dimensions of an image.
--	---------------------------------

euporie.output.container.get_dims

`euporie.output.container.get_dims(data: Any, format_: str, px: Optional[int] = None, py: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → tuple[Optional[int], Optional[float]]`

Get the dimensions of an image.

Foreground and background color are set at this point if they are available, as data conversion outputs are cached and re-used.

Parameters

- **data** – The data to check the dimensions of
- **format** – The current format of the data
- **px** – The desired pixel width of the data if known
- **py** – The pixel height of the data if known
- **fg** – The desired foreground color of the data
- **bg** – The desired background color of the data

Returns

A tuple of the data's width in terminal columns and its aspect ratio, when converted to a image.

Classes

<code>CellOutput(json[, height, width, focusable, ...])</code>	A container for rendered cell outputs.
<code>GraphicWindow(content, target_window, *args, ...)</code>	A window responsible for displaying terminal graphics content.
<code>OutputWindow(*args, **kwargs)</code>	A window sub-class which holds a reference to a graphic float.

euporie.output.container.CellOutput

```
class euporie.output.container.CellOutput(json: dict[str, Any], height: AnyDimension = None, width: AnyDimension = None, focusable: FilterOrBool = False, focus_on_click: FilterOrBool = False, show_scrollbar: FilterOrBool = False, wrap_lines: FilterOrBool = False, always_hide_cursor: FilterOrBool = True, style: Union[str, Callable[[], str]] = "")
```

A container for rendered cell outputs.

euporie.output.container.GraphicWindow

```
class euporie.output.container.GraphicWindow(content: OutputControl, target_window: Window, *args: Any, **kwargs: Any)
```

A window responsible for displaying terminal graphics content.

The graphic will be displayed if: - a completion menu is not being shown - a dialog is not being shown - a menu is not being shown - the output it attached to is fully in view

euporie.output.container.OutputWindow

```
class euporie.output.container.OutputWindow(*args: Any, **kwargs: Any)
```

A window sub-class which holds a reference to a graphic float.

```
class euporie.output.container.CellOutput(json: dict[str, Any], height: AnyDimension = None, width: AnyDimension = None, focusable: FilterOrBool = False, focus_on_click: FilterOrBool = False, show_scrollbar: FilterOrBool = False, wrap_lines: FilterOrBool = False, always_hide_cursor: FilterOrBool = True, style: Union[str, Callable[[], str]] = "")
```

Bases: `object`

A container for rendered cell outputs.

property data: `dict[str, str]`

Return dictionary of mime types and data for this output.

This generates similarly structured data objects for markdown cells and text output streams.

Returns JSON dictionary mapping mimes type to representation data.

```
class euporie.output.container.GraphicWindow(content: OutputControl, target_window: Window, *args: Any, **kwargs: Any)
```

Bases: `prompt_toolkit.layout.containers.Window`

A window responsible for displaying terminal graphics content.

The graphic will be displayed if: - a completion menu is not being shown - a dialog is not being shown - a menu is not being shown - the output it attached to is fully in view

content: `OutputControl`

get_children() → `List[prompt_toolkit.layout.containers.Container]`

Return the list of child `Container` objects.

get_key_bindings() → `Optional[prompt_toolkit.key_binding.key_bindings.KeyBindingsBase]`

Returns a `KeyBindings` object. These bindings become active when any user control in this container has the focus, except if any containers between this container and the focused user control is modal.

is_modal() → bool

When this container is modal, key bindings from parent containers are not taken into account if a user control in this container is focused.

preferred_height(width: int, max_available_height: int) → prompt_toolkit.layout.dimension.Dimension

Calculate the preferred height for this window.

preferred_width(max_available_width: int) → prompt_toolkit.layout.dimension.Dimension

Calculate the preferred width for this window.

reset() → None

Reset the state of this container and all the children. (E.g. reset scroll offsets, etc...)

write_to_screen(screen: Screen, mouse_handlers: MouseHandlers, write_position: WritePosition, parent_style: str, erase_bg: bool, z_index: Optional[int]) → None

Draws the graphic window's contents to the screen if required.

class euporie.output.container.**OutputWindow**(*args: Any, **kwargs: Any)

Bases: prompt_toolkit.layout.containers.Window

A window sub-class which holds a reference to a graphic float.

get_children() → List[prompt_toolkit.layout.containers.Container]

Return the list of child Container objects.

get_key_bindings() → Optional[prompt_toolkit.key_binding.key_bindings.KeyBindingsBase]

Returns a KeyBindings object. These bindings become active when any user control in this container has the focus, except if any containers between this container and the focused user control is modal.

is_modal() → bool

When this container is modal, key bindings from parent containers are not taken into account if a user control in this container is focused.

preferred_height(width: int, max_available_height: int) → prompt_toolkit.layout.dimension.Dimension

Calculate the preferred height for this window.

preferred_width(max_available_width: int) → prompt_toolkit.layout.dimension.Dimension

Calculate the preferred width for this window.

reset() → None

Reset the state of this container and all the children. (E.g. reset scroll offsets, etc...)

write_to_screen(screen: prompt_toolkit.layout.screen.Screen, mouse_handlers: prompt_toolkit.layout.mouse_handlers.MouseHandlers, write_position: prompt_toolkit.layout.screen.WritePosition, parent_style: str, erase_bg: bool, z_index: Optional[int]) → None

Write window to screen. This renders the user control, the margins and copies everything over to the absolute position at the given screen.

euporie.output.container.get_dims(data: Any, format_: str, px: Optional[int] = None, py: Optional[int] = None, fg: Optional[str] = None, bg: Optional[str] = None) → tuple[Optional[int], Optional[float]]

Get the dimensions of an image.

Foreground and background color are set at this point if they are available, as data conversion outputs are cached and re-used.

Parameters

- **data** – The data to check the dimensions of
- **format** – The current format of the data

- **px** – The desired pixel width of the data if known
- **py** – The pixel height of the data if known
- **fg** – The desired foreground color of the data
- **bg** – The desired background color of the data

Returns

A tuple of the data's width in terminal columns and its aspect ratio, when converted to a image.

euporie.output.control

Defines custom controls which re-render on resize.

Classes

<code>AnsiControl</code> (data, format_[, fg_color, ...])	A data formatter, which displays cell output data.
<code>ItermGraphicControl</code> (data, format_[, ...])	
<code>KittyGraphicControl</code> (data, format_[, ...])	
<code>OutputControl</code> (data, format_[, fg_color, ...])	A data formatter, which displays cell output data.
<code>SixelGraphicControl</code> (data, format_[, ...])	

euporie.output.control.AnsiControl

```
class euporie.output.control.AnsiControl(data: Any, format_: str, fg_color: Optional[str] = None,
                                          bg_color: Optional[str] = None, sizing_func:
                                          Optional[Callable] = None, focusable: FilterOrBool = False,
                                          focus_on_click: FilterOrBool = False)
```

A data formatter, which displays cell output data.

It will attempt to display the data in the best way possible, and reacts to resize events - i.e. images are downscaled to fit, markdown is re-flowed, etc.

euporie.output.control.ItemGraphicControl

```
class euporie.output.control.ItemGraphicControl(data: Any, format_: str, fg_color: Optional[str] =
None, bg_color: Optional[str] = None, sizing_func:
Optional[Callable] = None)
```

euporie.output.control.KittyGraphicControl

```
class euporie.output.control.KittyGraphicControl(data: Any, format_: str, fg_color: Optional[str] =
None, bg_color: Optional[str] = None, sizing_func:
Optional[Callable] = None)
```

euporie.output.control.OutputControl

```
class euporie.output.control.OutputControl(data: Any, format_: str, fg_color: Optional[str] = None,
bg_color: Optional[str] = None, sizing_func:
Optional[Callable] = None, focusable: FilterOrBool =
False, focus_on_click: FilterOrBool = False)
```

A data formatter, which displays cell output data.

It will attempt to display the data in the best way possible, and reacts to resize events - i.e. images are downscaled to fit, markdown is re-flowed, etc.

euporie.output.control.SixelGraphicControl

```
class euporie.output.control.SixelGraphicControl(data: Any, format_: str, fg_color: Optional[str] =
None, bg_color: Optional[str] = None, sizing_func:
Optional[Callable] = None, focusable: FilterOrBool =
False, focus_on_click: FilterOrBool = False)
```

```
class euporie.output.control.AnsiControl(data: Any, format_: str, fg_color: Optional[str] = None,
bg_color: Optional[str] = None, sizing_func:
Optional[Callable] = None, focusable: FilterOrBool = False,
focus_on_click: FilterOrBool = False)
```

Bases: [euporie.output.control.OutputControl](#)

A data formatter, which displays cell output data.

It will attempt to display the data in the best way possible, and reacts to resize events - i.e. images are downscaled to fit, markdown is re-flowed, etc.

property aspect: `Optional[float]`

Lazily load the aspect ratio of the output.

create_content(width: int, height: int) → [prompt_toolkit.layout.controls.UIContent](#)

Generates rendered output at a given size.

Parameters

- **width** – The desired output width
- **height** – The desired output height

Returns `UIContent` for the given output size.

property cursor_position: `prompt_toolkit.data_structures.Point`
Get the cursor position.

get_invalidate_events() \rightarrow `Iterable[Event[object]]`
Return the Window invalidate events.

get_key_bindings() \rightarrow `Optional[KeyBindingsBase]`
The key bindings that are specific for this user control.

Return a `KeyBindings` object if some key bindings are specified, or *None* otherwise.

get_rendered_lines(*width: int, height: int*) \rightarrow `list[StyleAndTextTuples]`
Get rendered lines from the cache, or generate them.

hide() \rightarrow `None`
Hides the output from show.

is_focusable() \rightarrow `bool`
Determines if the current control is focusable.

property max_cols: `int`
Lazily load the maximum width of the output in terminal columns.

mouse_handler(*mouse_event: MouseEvent*) \rightarrow `NotImplementedOrNone`
Mouse handler for this control.

move_cursor_down() \rightarrow `None`
Moves the cursor down one line.

move_cursor_up() \rightarrow `None`
Moves the cursor up one line.

preferred_height(*width: int, max_available_height: int, wrap_lines: bool, get_line_prefix: Optional[GetLinePrefixCallable]*) \rightarrow `int`
Returns the number of lines in the rendered content.

preferred_width(*max_available_width: int*) \rightarrow `Optional[int]`
Returns the width of the rendered content.

reset() \rightarrow `None`

size() \rightarrow `None`
Lazily load the maximum width and aspect ratio of the output.

5.8.20 euporie.palette

Contains the command palette container.

Classes

<code>CommandMenuControl</code> (<i>command_palette[, width]</i>)	A filterable and navigable list of commands.
<code>CommandPalette</code> ()	A command palette which allows searching the available commands.

euporie.palette.CommandMenuControl

class euporie.palette.**CommandMenuControl**(*command_palette: euporie.palette.CommandPalette, width: int = 60*)

A filterable and navigable list of commands.

euporie.palette.CommandPalette

class euporie.palette.**CommandPalette**

A command palette which allows searching the available commands.

class euporie.palette.**CommandMenuControl**(*command_palette: euporie.palette.CommandPalette, width: int = 60*)

Bases: `prompt_toolkit.layout.controls.UIControl`

A filterable and navigable list of commands.

create_content(*width: int, height: int*) → `prompt_toolkit.layout.controls.UIContent`
Create a `UIContent` object for this control.

get_invalidate_events() → `Iterable[Event[object]]`
Return a list of `Event` objects. This can be a generator. (The application collects all these events, in order to bind redraw handlers to these events.)

get_key_bindings() → `Optional[KeyBindingsBase]`
The key bindings that are specific for this user control.
Return a `KeyBindings` object if some key bindings are specified, or `None` otherwise.

is_focusable() → `bool`
Tell whether this user control is focusable.

mouse_handler(*mouse_event: MouseEvent*) → `NotImplementedOrNone`
Handle clicking and scrolling mouse events.

move_cursor_down() → `None`
Request to move the cursor down. This happens when scrolling down and the cursor is completely at the top.

move_cursor_up() → `None`
Request to move the cursor up.

preferred_height(*width: int, max_available_height: int, wrap_lines: bool, get_line_prefix: Optional[GetLinePrefixCallable]*) → `Optional[int]`
Returns the preferred height of the command list.

preferred_width(*max_available_width: int*) → `Optional[int]`
Returns the preferred width of the command list.

reset() → `None`

class euporie.palette.**CommandPalette**

Bases: `object`

A command palette which allows searching the available commands.

accept(*buffer: Optional[Buffer] = None*) → `bool`
Called on enter: runs the selected command.

hide(*event: KeyPressEvent = None*) → `None`
Hides the command palette and returns focus to what was focused before.

index: `int`

matches: `list[_CommandMatch]`

select(*n: int, event: KeyPressEvent = None*) → `None`
Change the index of the selected command.

Parameters

- **n** – The relative amount by which to change the selected index
- **event** – Ignored

show() → `None`

Displays and focuses the command palette.

statusbar_fields() → `StatusBarFields`

Returns a list of statusbar field values shown then this tab is active.

text_changed(*buffer: Buffer*) → `None`

Called when the input text changes: filters the command list.

toggle() → `None`

Shows or hides the command palette.

5.8.21 euporie.scroll

Contains the *ScrollingContainer* class, which renders children on the fly.

Classes

<code>ChildRenderInfo</code> (parent, child)	A class which holds information about a <i>ScrollingContainer</i> child.
<code>ScrollbarControl</code> (target)	A control which displays a <code>ScrollbarMargin</code> for another container.
<code>ScrollingContainer</code> (children_func[, height, ...])	A scrollable container which renders only the currently visible children.

euporie.scroll.ChildRenderInfo

class `euporie.scroll.ChildRenderInfo`(*parent: ScrollingContainer, child: Cell*)

A class which holds information about a *ScrollingContainer* child.

euporie.scroll.ScrollbarControl

class euporie.scroll.**ScrollbarControl**(*target: euporie.scroll.ScrollingContainer*)
 A control which displays a ScrollbarMargin for another container.

euporie.scroll.ScrollingContainer

class euporie.scroll.**ScrollingContainer**(*children_func: Callable, height: AnyDimension = None, width: AnyDimension = None, style: Union[str, Callable[[], str]] = ""*)
 A scrollable container which renders only the currently visible children.

class euporie.scroll.**ChildRenderInfo**(*parent: ScrollingContainer, child: Cell*)
 Bases: **object**

A class which holds information about a *ScrollingContainer* child.

blit(*screen: prompt_toolkit.layout.screen.Screen, mouse_handlers: prompt_toolkit.layout.mouse_handlers.MouseHandlers, left: int, top: int, cols: slice, rows: slice*) → *None*

Copies the rendered child from the local screen to the main screen.

All locations are adjusted, allowing the pre-rendered child to be placed at any location on the main screen.

Parameters

- **screen** – The main screen to copy the pre-rendered screen data to
- **mouse_handlers** – The mouse handler collection to copy the pre-rendered handlers to
- **left** – The left-most column in which to start placing the data
- **top** – The upper row in which to start placing the data
- **cols** – The number of columns top copy
- **rows** – The number of rows to copy

render(*available_width: int, available_height: int, style: str = ""*) → *None*
 Renders the child container at a given size.

Parameters

- **available_width** – The height available for rendering
- **available_height** – The width available for rendering
- **style** – The parent style to apply when rendering

class euporie.scroll.**ScrollbarControl**(*target: euporie.scroll.ScrollingContainer*)
 Bases: *prompt_toolkit.layout.controls.UIControl*

A control which displays a ScrollbarMargin for another container.

create_content(*width: int, height: int*) → *prompt_toolkit.layout.controls.UIContent*
 Generates the scrollbar's content.

get_invalidate_events() → *Iterable[Event[object]]*
 Return a list of *Event* objects. This can be a generator. (The application collects all these events, in order to bind redraw handlers to these events.)

get_key_bindings() → *Optional[KeyBindingsBase]*
 The key bindings that are specific for this user control.

Return a *KeyBindings* object if some key bindings are specified, or *None* otherwise.

is_focusable() → *bool*

Tell whether this user control is focusable.

mouse_handler(*mouse_event: MouseEvent*) → *NotImplementedOrNone*

Handle mouse events.

move_cursor_down() → *None*

Request to move the cursor down. This happens when scrolling down and the cursor is completely at the top.

move_cursor_up() → *None*

Request to move the cursor up.

preferred_height(*width: int, max_available_height: int, wrap_lines: bool, get_line_prefix: Optional[Callable[[int, int], AnyFormattedText]]*) → *Optional[int]*

Get the preferred height of the scrollbar: all of the height available.

preferred_width(*max_available_width: int*) → *int*

Return the preferred width of this scrollbar.

reset() → *None*

class *euporie.scroll.ScrollingContainer*(*children_func: Callable, height: AnyDimension = None, width: AnyDimension = None, style: Union[str, Callable[[], str]] = ""*)

Bases: *prompt_toolkit.layout.containers.Container*

A scrollable container which renders only the currently visible children.

property children: *Sequence[Cell]*

Return the current children of this container instance.

get_child(*index: Optional[int] = None*) → *AnyContainer*

Return a rendered instance of the child at the given index.

If no index is given, the currently selected child is returned.

Parameters *index* – The index of the child to return.

Returns A rendered instance of the child.

get_child_render_info(*index: Optional[int] = None*) → *ChildRenderInfo*

Return a rendered instance of the child at the given index.

If no index is given, the currently selected child is returned.

Parameters *index* – The index of the child to return.

Returns A rendered instance of the child.

get_children() → *List['Container']*

Return the list of currently visible children to include in the layout.

get_key_bindings() → *Optional[prompt_toolkit.key_binding.key_bindings.KeyBindingsBase]*

Returns a *KeyBindings* object. These bindings become active when any user control in this container has the focus, except if any containers between this container and the focused user control is modal.

is_modal() → *bool*

When this container is modal, key bindings from parent containers are not taken into account if a user control in this container is focused.

property known_sizes: *dict[int, int]*

A dictionary mapping child indices to height values.

mouse_scroll_handler(*mouse_event: prompt_toolkit.mouse_events.MouseEvent*) → *None*

A mouse handler to scroll the pane.

preferred_height(*width: int, max_available_height: int*) → `prompt_toolkit.layout.dimension.Dimension`

Return the preferred height only if one is provided.

preferred_width(*max_available_width: int*) → `prompt_toolkit.layout.dimension.Dimension`

Do not provide a preferred width - grow to fill the available space.

reset() → `None`

Reset the state of this container and all the children.

scroll(*n: int*) → `None`

Scrolls up or down a number of rows.

Parameters *n* – The number of rows to scroll, negative for up, positive for down

scroll_to(*index: int*) → `None`

Scroll a child into view.

Parameters *index* – The child index to scroll into view

property selected_indices: `list[int]`

Returns in indices of the currently selected children.

property selected_slice: `slice`

Returns the currently selected slice, checking for changes in focus.

validate_slice(*slice_: slice*) → `slice`

Ensures a slice describes a valid range of children.

write_to_screen(*screen: Screen, mouse_handlers: MouseHandlers, write_position: WritePosition, parent_style: str, erase_bg: bool, z_index: Optional[int]*) → `None`

Write the actual content to the screen.

Children are rendered only if they are visible and have changed, and the output is cached to a separate screen object stored in a `py:ChildRenderInfo`. The cached rendering of the children which are actually visible are then copied to the screen. This results in faster rendering of the scrolling container, and makes scrolling more performant.

Parameters

- **screen** – The `Screen` class to which the output has to be written.
- **mouse_handlers** – `prompt_toolkit.layout.mouse_handlers.MouseHandlers`.
- **write_position** – A `prompt_toolkit.layout.screen.WritePosition` object defining where this container should be drawn.
- **erase_bg** – If true, the background will be erased prior to drawing.
- **parent_style** – Style string to pass to the Window object. This will be applied to all content of the windows. `VSplit` and `prompt_toolkit.layout.containers.HSplit` can use it to pass their style down to the windows that they contain.
- **z_index** – Used for propagating `z_index` from parent to child.

5.8.22 euporie.style

Style related functions.

Functions

<code>color_series([n, interval])</code>	Create a series of dimmed colours.
--	------------------------------------

euporie.style.color_series

`euporie.style.color_series(n: int = 6, interval: float = 0.05, **kwargs: Any) → dict`
Create a series of dimmed colours.

`euporie.style.color_series(n: int = 6, interval: float = 0.05, **kwargs: Any) → dict`
Create a series of dimmed colours.

5.8.23 euporie.suggest

Suggests line completions from kernel history.

Classes

<code>AppendLineAutoSuggestion([style])</code>	Append the auto suggestion to the current line of the input.
<code>ConditionalAutoSuggestAsync(auto_suggest, filter)</code>	Auto suggest that can be turned on and of according to a certain condition.
<code>KernelAutoSuggest(kernel)</code>	Suggests line completions from kernel history.

euporie.suggest.AppendLineAutoSuggestion

`class euporie.suggest.AppendLineAutoSuggestion(style: str = 'class:auto-suggestion')`
Append the auto suggestion to the current line of the input.

euporie.suggest.ConditionalAutoSuggestAsync

```
class euporie.suggest.ConditionalAutoSuggestAsync(auto_suggest: AutoSuggest, filter: Union[bool, Filter])
```

Auto suggest that can be turned on and of according to a certain condition.

euporie.suggest.KernelAutoSuggest

```
class euporie.suggest.KernelAutoSuggest(kernel: NotebookKernel)
```

Suggests line completions from kernel history.

```
class euporie.suggest.KernelAutoSuggest(kernel: NotebookKernel)
```

Bases: `prompt_toolkit.auto_suggest.AutoSuggest`

Suggests line completions from kernel history.

```
get_suggestion(buffer: Buffer, document: Document) → Optional[Suggestion]
```

Does nothing.

```
async get_suggestion_async(buff: Buffer, document: Document) → Optional[Suggestion]
```

Return suggestions based on matching kernel history.

5.8.24 euporie.tab

Contains the main class for a notebook file.

Classes

<code>Tab()</code>	Base class for interface tabs.
--------------------	--------------------------------

euporie.tab.Tab

```
class euporie.tab.Tab
```

Base class for interface tabs.

```
class euporie.tab.Tab
```

Bases: `object`

Base class for interface tabs.

```
close(cb: Optional[Callable]) → None
```

Function to close a tab with a callback.

Parameters **cb** – A function to call after the tab is closed.

```
container: _Split
```

```
focus() → None
```

Focuses the tab.

```
statusbar_fields() → tuple[Sequence[AnyFormattedText], Sequence[AnyFormattedText]]
```

Returns a list of statusbar field values shown then this tab is active.

```
property title: str
```

Return the tab title.

5.8.25 euporie.terminal

Contains classes related to querying terminal features.

Functions

<code>tmuxify(cmd)</code>	Wraps an escape sequence for tmux passthrough.
---------------------------	--

euporie.terminal.tmuxify

`euporie.terminal.tmuxify(cmd: str) → str`
Wraps an escape sequence for tmux passthrough.

Classes

<code>BackgroundColor(output)</code>	A terminal query to check the terminal's background colour.
<code>ColorQueryMixin()</code>	A mixin for terminal queries which check a terminal colour.
<code>DepthOfColor(output)</code>	Determines the suspected color depth of the terminal.
<code>ForegroundColor(output)</code>	A terminal query to check the terminal's foreground colour.
<code>ItemGraphicsStatus(output)</code>	A terminal query to check for item graphics support.
<code>KittyGraphicsStatus(output)</code>	A terminal query to check for kitty graphics support.
<code>PixelDimensions(output)</code>	A terminal query to check the terminal's dimensions in pixels.
<code>SixelGraphicsStatus(output)</code>	A terminal query to check for sixel graphics support.
<code>TerminalInfo(input_, output)</code>	A class to gather and hold information about the terminal.
<code>TerminalQuery(output)</code>	A class representing a terminal query.
<code>Vt100Parser(*args, **kwargs)</code>	A Vt100Parser which checks input against additional key patterns.

euporie.terminal.BackgroundColor

class `euporie.terminal.BackgroundColor(output: Output)`
A terminal query to check the terminal's background colour.

euporie.terminal.ColorQueryMixin

class `euporie.terminal.ColorQueryMixin`
A mixin for terminal queries which check a terminal colour.

euporie.terminal.DepthOfColor

class euporie.terminal.**DepthOfColor**(*output: Output*)
Determines the suspected color depth of the terminal.

euporie.terminal.ForegroundColor

class euporie.terminal.**ForegroundColor**(*output: Output*)
A terminal query to check the terminal's foreground colour.

euporie.terminal.ItemGraphicsStatus

class euporie.terminal.**ItemGraphicsStatus**(*output: Output*)
A terminal query to check for item graphics support.

euporie.terminal.KittyGraphicsStatus

class euporie.terminal.**KittyGraphicsStatus**(*output: Output*)
A terminal query to check for kitty graphics support.

euporie.terminal.PixelDimensions

class euporie.terminal.**PixelDimensions**(*output: Output*)
A terminal query to check the terminal's dimensions in pixels.

euporie.terminal.SixelGraphicsStatus

class euporie.terminal.**SixelGraphicsStatus**(*output: Output*)
A terminal query to check for sixel graphics support.

euporie.terminal.TerminalInfo

class euporie.terminal.**TerminalInfo**(*input_: Input, output: Output*)
A class to gather and hold information about the terminal.

euporie.terminal.TerminalQuery

class euporie.terminal.**TerminalQuery**(*output: Output*)
A class representing a terminal query.

This allows a control sequence to sent to the terminal, the response interpreted, and the recieved value processed and stored.

euporie.terminal.Vt100Parser**class** euporie.terminal.Vt100Parser(*args: Any, **kwargs: Any)

A Vt100Parser which checks input against additional key patterns.

class euporie.terminal.BackgroundColor(output: Output)Bases: [euporie.terminal.ColorQueryMixin](#), [euporie.terminal.TerminalQuery](#)

A terminal query to check the terminal's background colour.

cache = True**cmd** = '\x1b]10;?\x1b\\'**default:** Optional[Any] = '#000000'**pattern:** re.Pattern = re.compile('^\\x1b\\]11;rgb:(?P<r>[0-9A-Fa-f]{2,4})/(?P<g>[0-9A-Fa-f]{2,4})/(?P[0-9A-Fa-f]{2,4})\\x1b\\\\\\\\Z')**send()** → None

Sends the terminal query command to the output.

property value: Any

Returns the last known value for the query.

Returns The last value recieved, or the default value.**verify(data: str)** → Optional[str]

Verifies the response contains a colour.

class euporie.terminal.ColorQueryMixinBases: [object](#)

A mixin for terminal queries which check a terminal colour.

pattern: re.Pattern**verify(data: str)** → Optional[str]

Verifies the response contains a colour.

class euporie.terminal.DepthOfColor(output: Output)Bases: [euporie.terminal.TerminalQuery](#)

Determines the suspected color depth of the terminal.

cache = False**cmd** = ''**default:** Optional[Any] = 'DEPTH_24_BIT'**pattern:** Optional[re.Pattern] = None**send()** → None

Sends the terminal query command to the output.

property value: Any

Returns the last known value for the query.

Returns The last value recieved, or the default value.**verify(data: str)** → Optional[Any]

Verifies the response from the terminal.

```

class euporie.terminal.ForegroundColor(output: Output)
  Bases: euporie.terminal.ColorQueryMixin, euporie.terminal.TerminalQuery
  A terminal query to check the terminal's foreground colour.

  cache = True
  cmd = '\x1b]11;?\x1b\\'
  default: Optional[Any] = '#FFFFFF'
  pattern: re.Pattern = re.compile('^\\x1b\\]10;rgb:(?P<r>[0-9A-Fa-f]{2,4})/(?P<g>[0-9A-Fa-f]{2,4})/(?P<b>[0-9A-Fa-f]{2,4})\\x1b\\\\\\\\Z')
  send() → None
    Sends the terminal query command to the output.

  property value: Any
    Returns the last known value for the query.

    Returns The last value recieved, or the default value.

  verify(data: str) → Optional[str]
    Verifies the response contains a colour.

class euporie.terminal.ItemGraphicsStatus(output: Output)
  Bases: euporie.terminal.TerminalQuery
  A terminal query to check for item graphics support.

  cache = True
  cmd = ''
  default: Optional[Any] = False
  pattern: Optional[re.Pattern] = None
  send() → None
    Sends the terminal query command to the output.

  property value: Any
    Returns the last known value for the query.

    Returns The last value recieved, or the default value.

  verify(data: str) → Optional[Any]
    Verifies the response from the terminal.

class euporie.terminal.KittyGraphicsStatus(output: Output)
  Bases: euporie.terminal.TerminalQuery
  A terminal query to check for kitty graphics support.

  cache = True
  cmd = '\x1b_Gi=4294967295,s=1,v=1,a=q,t=d,f=24;AAAA\x1b\\'
  default: Optional[Any] = False
  pattern: Optional[re.Pattern] =
    re.compile('^\\x1b_Gi=4294967295;(P<status>OK)\\x1b\\\\\\\\Z')
  send() → None
    Sends the terminal query command to the output.

```

property value: Any

Returns the last known value for the query.

Returns The last value recieved, or the default value.

verify(data: str) → bool

Verifies the terminal response means kitty graphics are supported.

class euporie.terminal.PixelDimensions(output: Output)

Bases: [euporie.terminal.TerminalQuery](#)

A terminal query to check the terminal's dimensions in pixels.

cache = True

cmd = '\x1b[14t'

default: Optional[Any] = (0, 0)

pattern: Optional[re.Pattern] =

re.compile('^\x1b\[4;(?P<y>\\d+);(?P<x>\\d+)t\\Z')

send() → None

Sends the terminal query command to the output.

property value: Any

Returns the last known value for the query.

Returns The last value recieved, or the default value.

verify(data: str) → Optional[tuple[int, int]]

Verifies the terminal responded with pixel dimensions.

class euporie.terminal.SixelGraphicsStatus(output: Output)

Bases: [euporie.terminal.TerminalQuery](#)

A terminal query to check for sixel graphics support.

cache = True

cmd = '\x1b[c'

default: Optional[Any] = False

pattern: Optional[re.Pattern] =

re.compile('^\x1b\[\\[\\?(?:\\d+;)*(?P<sixel>4)(?:;\\d+)*c\\Z')

send() → None

Sends the terminal query command to the output.

property value: Any

Returns the last known value for the query.

Returns The last value recieved, or the default value.

verify(data: str) → bool

Verifies the terminal response means sixel graphics are supported.

class euporie.terminal.TerminalInfo(input_: Input, output: Output)

Bases: [object](#)

A class to gather and hold information about the terminal.

property cell_size_px: tuple[int, int]

Get the pixel size of a single terminal cell.

input: Input

output: `Output`

register(*query: Type[TerminalQuery]*) → *TerminalQuery*

Instantiates and registers a query's response with the input parser.

send_all() → `None`

Sends the command for all queries.

property terminal_size_px: `tuple[int, int]`

Get the pixel dimensions of the terminal.

class `euporie.terminal.TerminalQuery(output: Output)`

Bases: `object`

A class representing a terminal query.

This allows a control sequence to sent to the terminal, the response interpreted, and the recieved value processed and stored.

cache = `False`

cmd = `''`

default: `Optional[Any] = None`

pattern: `Optional[re.Pattern] = None`

send() → `None`

Sends the terminal query command to the output.

property value: `Any`

Returns the last known value for the query.

Returns The last value recieved, or the default value.

verify(*data: str*) → `Optional[Any]`

Verifies the response from the terminal.

class `euporie.terminal.Vt100Parser(*args: Any, **kwargs: Any)`

Bases: `prompt_toolkit.input.vt100_parser.Vt100Parser`

A Vt100Parser which checks input against additional key patterns.

feed(*data: str*) → `None`

Feed the input stream.

Parameters data – Input string (unicode).

feed_and_flush(*data: str*) → `None`

Wrapper around `feed` and `flush`.

flush() → `None`

Flush the buffer of the input stream.

This will allow us to handle the escape key (or maybe meta) sooner. The input received by the escape key is actually the same as the first characters of e.g. Arrow-Up, so without knowing what follows the escape sequence, we don't know whether escape has been pressed, or whether it's something else. This flush function should be called after a timeout, and processes everything that's still in the buffer as-is, so without assuming any characters will follow.

reset(*request: bool = False*) → `None`

`euporie.terminal.tmuxify(cmd: str)` → `str`

Wraps an escape sequence for tmux passthrough.

5.8.26 euporie.text

Contains dputed ANSI parsing and Formatted Text processing.

Classes

<code>ANSI(value)</code>	Converts ANSI text into formatted text, preserving all control sequences.
<code>FormatTextProcessor(formatted_text)</code>	Applies formatted text to a TextArea.
<code>FormattedTextArea(formatted_text, *args, ...)</code>	Applies formatted text to a TextArea.

euporie.text.ANSI

class euporie.text.ANSI(*value: str*)
Converts ANSI text into formatted text, preserving all control sequences.

euporie.text.FormatTextProcessor

class euporie.text.FormatTextProcessor(*formatted_text: StyleAndTextTuples*)
Applies formatted text to a TextArea.

euporie.text.FormattedTextArea

class euporie.text.FormattedTextArea(*formatted_text: AnyFormattedText, *args: Any, **kwargs: Any*)
Applies formatted text to a TextArea.

class euporie.text.ANSI(*value: str*)
Bases: `prompt_toolkit.formatted_text.ansi.ANSI`
Converts ANSI text into formatted text, preserving all control sequences.
format(**args: str, **kwargs: str*) → `prompt_toolkit.formatted_text.ansi.ANSI`
Like *str.format*, but make sure that the arguments are properly escaped. (No ANSI escapes can be injected.)

class euporie.text.FormatTextProcessor(*formatted_text: StyleAndTextTuples*)
Bases: `prompt_toolkit.layout.processors.Processor`
Applies formatted text to a TextArea.
apply_transformation(*transformation_input: TransformationInput*) → Transformation
Apply text formatting to a line in a buffer.

class euporie.text.FormattedTextArea(*formatted_text: AnyFormattedText, *args: Any, **kwargs: Any*)
Bases: `prompt_toolkit.widgets.base.TextArea`
Applies formatted text to a TextArea.
property accept_handler: Optional[Callable[[`prompt_toolkit.buffer.Buffer`], bool]]
The accept handler. Called when the user accepts the input.
property document: `prompt_toolkit.document.Document`
The *Buffer* document (text + cursor position).
property formatted_text: StyleAndTextTuples
The formatted text.

get_processor() → *euporie.text.FormatTextProcessor*
Generate a processor for the formatted text.

property text: **str**
The *Buffer* text.

INDICES AND TABLES

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

PYTHON MODULE INDEX

e

- euporie, 36
- euporie.app, 37
- euporie.app.base, 38
- euporie.app.current, 42
- euporie.app.dump, 43
- euporie.app.tui, 48
- euporie.box, 54
- euporie.cell, 57
- euporie.commands, 63
- euporie.commands.base, 63
- euporie.commands.buffer, 65
- euporie.commands.cell, 73
- euporie.commands.cell_output, 74
- euporie.commands.completions, 75
- euporie.commands.config, 76
- euporie.commands.format, 79
- euporie.commands.notebook, 80
- euporie.commands.pager, 87
- euporie.commands.registry, 87
- euporie.commands.search, 88
- euporie.commands.suggestions, 88
- euporie.commands.tui, 89
- euporie.completion, 92
- euporie.config, 93
- euporie.containers, 95
- euporie.convert, 96
- euporie.convert.base, 96
- euporie.convert.formats, 98
- euporie.convert.formats.ansi, 98
- euporie.convert.formats.base64, 104
- euporie.convert.formats.common, 104
- euporie.convert.formats.jpeg, 105
- euporie.convert.formats.markdown, 105
- euporie.convert.formats.pdf, 106
- euporie.convert.formats.pil, 106
- euporie.convert.formats.png, 107
- euporie.convert.formats.rich, 108
- euporie.convert.formats.sixel, 108
- euporie.convert.util, 109
- euporie.filters, 110
- euporie.format, 110

- euporie.kernel, 111
- euporie.key_binding, 115
- euporie.key_binding.bindings, 116
- euporie.key_binding.bindings.commands, 116
- euporie.key_binding.bindings.micro, 116
- euporie.key_binding.micro_state, 117
- euporie.key_binding.util, 118
- euporie.keys, 118
- euporie.log, 120
- euporie.margins, 123
- euporie.markdown, 124
- euporie.markdown.blocks, 125
- euporie.markdown.blocks.math, 125
- euporie.markdown.blocks.tables, 125
- euporie.markdown.inlines, 127
- euporie.markdown.parser, 129
- euporie.markdown.rich, 131
- euporie.menu, 141
- euporie.menu.bar, 141
- euporie.menu.contents, 142
- euporie.menu.item, 142
- euporie.notebook, 144
- euporie.output, 146
- euporie.output.container, 147
- euporie.output.control, 150
- euporie.palette, 152
- euporie.scroll, 154
- euporie.style, 158
- euporie.suggest, 158
- euporie.tab, 159
- euporie.terminal, 160
- euporie.text, 166

Symbols

- v
 - command line option, 17
- autocomplete
 - command line option, 18
- autoformat
 - command line option, 18
- autoinspect
 - command line option, 18
- autosuggest
 - command line option, 18
- background-character
 - command line option, 18
- background-color
 - command line option, 18
- background-pattern
 - command line option, 18
- bg-char
 - command line option, 18
- bg-color
 - command line option, 18
- bg-pattern
 - command line option, 18
- color-scheme
 - command line option, 18
- debug
 - command line option, 17
- dump
 - command line option, 17
- dump-file
 - command line option, 17
- edit-mode
 - command line option, 18
- expand
 - command line option, 18
- format-black
 - command line option, 18
- format-isort
 - command line option, 18
- format-ssort
 - command line option, 18
- help
 - command line option, 17
- line-numbers
 - command line option, 18
- log-file
 - command line option, 17
- max-notebook-width
 - command line option, 18
- no-autocomplete
 - command line option, 18
- no-autoformat
 - command line option, 18
- no-autoinspect
 - command line option, 18
- no-autosuggest
 - command line option, 18
- no-debug
 - command line option, 17
- no-dump
 - command line option, 17
- no-expand
 - command line option, 18
- no-format-black
 - command line option, 18
- no-format-isort
 - command line option, 18
- no-format-ssort
 - command line option, 18
- no-line-numbers
 - command line option, 18
- no-page
 - command line option, 17
- no-run
 - command line option, 17
- no-show-cell-borders
 - command line option, 18
- no-show-status-bar
 - command line option, 18
- no-tmux-graphics
 - command line option, 18
- page
 - command line option, 17
- run

command line option, 17
--run-after-external-edit
command line option, 18
--show-cell-borders
command line option, 18
--show-status-bar
command line option, 18
--syntax-theme
command line option, 19
--tab-size
command line option, 18
--terminal-polling-interval
command line option, 18
--tmux-graphics
command line option, 18
--version
command line option, 17
-h
command line option, 17
<Path
command line option, 17

A

about() (in module *euporie.commands.tui*), 89, 91
accept() (*euporie.palette.CommandPalette* method), 153
accept_completion() (in module *euporie.commands.completions*), 75
accept_handler (*euporie.cell.CellInputTextArea* property), 60
accept_handler (*euporie.cell.CellStdinTextArea* property), 61
accept_handler (*euporie.text.FormattedTextArea* property), 166
accept_suggestion() (in module *euporie.commands.suggestions*), 88
accepts_lines (*euporie.markdown.blocks.math.MathBlock* attribute), 125
accepts_lines (*euporie.markdown.blocks.MathBlock* attribute), 127
accepts_lines (*euporie.markdown.blocks.Table* attribute), 127
accepts_lines (*euporie.markdown.blocks.tables.Table* attribute), 126
acquire() (*euporie.log.QueueHandler* method), 121
add() (*euporie.keys.KeyBindingsInfo* method), 118
add() (*euporie.notebook.Notebook* method), 145
add() (in module *euporie.commands*), 92
add() (in module *euporie.commands.base*), 63, 65
add() (in module *euporie.commands.registry*), 88
add_binding() (*euporie.keys.KeyBindingsInfo* method), 119
add_cell_above() (in module *euporie.commands.notebook*), 81, 85

add_cell_below() (in module *euporie.commands.notebook*), 81, 85
add_child() (*euporie.markdown.Parser* method), 139
add_child() (*euporie.markdown.parser.Parser* method), 130
add_float() (*euporie.app.base.EuporieApp* method), 38
add_float() (*euporie.app.dump.DumpApp* method), 43
add_float() (*euporie.app.tui.TuiApp* method), 48
add_keys() (*euporie.commands.base.Command* method), 64
add_keys() (*euporie.commands.Command* method), 91
add_line() (*euporie.markdown.Parser* method), 139
add_line() (*euporie.markdown.parser.Parser* method), 130
add_record() (*euporie.log.LogView* method), 121
addBracket() (*euporie.markdown.InlineParser* method), 135
addBracket() (*euporie.markdown.inlines.InlineParser* method), 128
addFilter() (*euporie.log.QueueHandler* method), 122
advance_next_nonspace() (*euporie.markdown.Parser* method), 139
advance_next_nonspace() (*euporie.markdown.parser.Parser* method), 130
advance_offset() (*euporie.markdown.Parser* method), 139
advance_offset() (*euporie.markdown.parser.Parser* method), 130
ANSI (class in *euporie.text*), 166
AnsiControl (class in *euporie.output.control*), 150, 151
AppendLineAutoSuggestion (class in *euporie.suggest*), 158
apply_transformation() (*euporie.text.FormatTextProcessor* method), 166
ask_file() (*euporie.app.tui.TuiApp* method), 48
ask_new_file() (*euporie.app.tui.TuiApp* method), 49
ask_open_file() (*euporie.app.tui.TuiApp* method), 49
aspect (*euporie.output.control.AnsiControl* property), 151
atx_heading() (*euporie.markdown.blocks.BlockStarts* static method), 126
autocomplete
command line option, 23
autocomplete() (in module *euporie.commands.config*), 76, 78
autofmt
command line option, 23
autofmt() (in module *euporie.commands.config*), 76, 78
autoinspect
command line option, 24

- autoinspect() (in module *euporie.commands.config*), 77, 78
 autosuggest
 command line option, 23
 autosuggest() (in module *euporie.commands.config*), 77, 78
 await_iopub_rsps() (*euporie.kernel.NotebookKernel* method), 112
 await_rsps() (*euporie.kernel.NotebookKernel* method), 112
 await_shell_rsps() (*euporie.kernel.NotebookKernel* method), 112
 await_stdin_rsps() (*euporie.kernel.NotebookKernel* method), 112
- ## B
- background_character
 command line option, 25
 background_color
 command line option, 25
 background_pattern
 command line option, 25
 BackgroundColor (class in *euporie.terminal*), 160, 162
 base64_to_bytes_py() (in module *euporie.convert.formats.common*), 104, 105
 bind() (*euporie.commands.base.Command* method), 64
 bind() (*euporie.commands.Command* method), 91
 bindings (*euporie.keys.KeyBindingsInfo* property), 119
 blit() (*euporie.scroll.ChildRenderInfo* method), 155
 block_quote() (*euporie.markdown.blocks.BlockStarts* static method), 126
 blocks (*euporie.markdown.Parser* attribute), 139
 blocks (*euporie.markdown.parser.Parser* attribute), 130
 BlockStarts (class in *euporie.markdown.blocks*), 126
 BooleanOptionalAction (class in *euporie.config*), 93
 border_char() (*euporie.cell.Cell* method), 59
 border_char() (*euporie.cell.InteractiveCell* method), 61
 border_style() (*euporie.cell.Cell* method), 59
 border_style() (*euporie.cell.InteractiveCell* method), 61
 BorderLine (class in *euporie.box*), 54, 55
 BOTTOM_LEFT (*euporie.box.RoundBorder* attribute), 56
 BOTTOM_LEFT (*euporie.box.SquareBorder* attribute), 57
 BOTTOM_RIGHT (*euporie.box.RoundBorder* attribute), 56
 BOTTOM_RIGHT (*euporie.box.SquareBorder* attribute), 57
 bytes_to_base64_py() (in module *euporie.convert.formats.base64*), 104
- ## C
- cache (*euporie.terminal.BackgroundColor* attribute), 162
 cache (*euporie.terminal.DepthOfColor* attribute), 162
 cache (*euporie.terminal.ForegroundColor* attribute), 163
 cache (*euporie.terminal.ItemGraphicsStatus* attribute), 163
 cache (*euporie.terminal.KittyGraphicsStatus* attribute), 163
 cache (*euporie.terminal.PixelDimensions* attribute), 164
 cache (*euporie.terminal.SixelGraphicsStatus* attribute), 164
 cache (*euporie.terminal.TerminalQuery* attribute), 165
 call_subproc() (in module *euporie.convert.util*), 109, 110
 can_contain() (*euporie.markdown.blocks.math.MathBlock* static method), 125
 can_contain() (*euporie.markdown.blocks.MathBlock* static method), 127
 can_contain() (*euporie.markdown.blocks.Table* static method), 127
 can_contain() (*euporie.markdown.blocks.tables.Table* static method), 126
 cancel_and_wait_for_background_tasks() (*euporie.app.base.EuporieApp* method), 38
 cancel_and_wait_for_background_tasks() (*euporie.app.dump.DumpApp* method), 43
 cancel_and_wait_for_background_tasks() (*euporie.app.tui.TuiApp* method), 49
 cancel_completion() (in module *euporie.commands.completions*), 75
 cancel_selection() (in module *euporie.commands.buffer*), 66, 71
 Cell (class in *euporie.cell*), 58, 59
 cell (*euporie.app.base.EuporieApp* property), 38
 cell (*euporie.app.dump.DumpApp* property), 43
 cell (*euporie.app.tui.TuiApp* property), 49
 cell_size_px (*euporie.terminal.TerminalInfo* property), 164
 cell_type (*euporie.cell.Cell* property), 59
 cell_type (*euporie.cell.InteractiveCell* property), 61
 CellInputTextArea (class in *euporie.cell*), 58, 60
 CellOutput (class in *euporie.output.container*), 148
 cells_to_code() (in module *euporie.commands.notebook*), 81, 85
 cells_to_markdown() (in module *euporie.commands.notebook*), 81, 85
 cells_to_raw() (in module *euporie.commands.notebook*), 81, 85
 CellStdinTextArea (class in *euporie.cell*), 58, 60
 chafa_convert() (in module *euporie.convert.formats.common*), 105
 change() (*euporie.kernel.NotebookKernel* method), 112
 change_kernel() (in module *euporie.commands.notebook*), 82, 85
 children (*euporie.containers.PrintingContainer* property), 95
 children (*euporie.scroll.ScrollingContainer* property), 156

`ChildRenderInfo` (class in `euporie.scroll`), 154, 155
`choices()` (`euporie.config.Config` method), 93
`cleanup_closed_tab()` (`euporie.app.base.EuporieApp` method), 38
`cleanup_closed_tab()` (`euporie.app.dump.DumpApp` method), 44
`cleanup_closed_tab()` (`euporie.app.tui.TuiApp` method), 49
`clear_output()` (`euporie.cell.Cell` method), 59
`clear_output()` (`euporie.cell.InteractiveCell` method), 61
`ClickArea` (class in `euporie.cell`), 59, 61
`clipboard` (`euporie.app.tui.TuiApp` attribute), 49
`close()` (`euporie.log.LogView` method), 121
`close()` (`euporie.log.QueueHandler` method), 122
`close()` (`euporie.notebook.Notebook` method), 145
`close()` (`euporie.tab.Tab` method), 159
`close_file()` (in module `euporie.commands.tui`), 89, 91
`close_pager()` (in module `euporie.commands.pager`), 87
`close_tab()` (`euporie.app.base.EuporieApp` method), 38
`close_tab()` (`euporie.app.dump.DumpApp` method), 44
`close_tab()` (`euporie.app.tui.TuiApp` method), 49
`close_unmatched_blocks()` (`euporie.markdown.Parser` method), 139
`close_unmatched_blocks()` (`euporie.markdown.parser.Parser` method), 130
`cmd` (`euporie.terminal.BackgroundColor` attribute), 162
`cmd` (`euporie.terminal.DepthOfColor` attribute), 162
`cmd` (`euporie.terminal.ForegroundColor` attribute), 163
`cmd` (`euporie.terminal.ItermGraphicsStatus` attribute), 163
`cmd` (`euporie.terminal.KittyGraphicsStatus` attribute), 163
`cmd` (`euporie.terminal.PixelDimensions` attribute), 164
`cmd` (`euporie.terminal.SixelGraphicsStatus` attribute), 164
`cmd` (`euporie.terminal.TerminalQuery` attribute), 165
`color_depth` (`euporie.app.base.EuporieApp` property), 38
`color_depth` (`euporie.app.dump.DumpApp` property), 44
`color_depth` (`euporie.app.tui.TuiApp` property), 49
`color_scheme`
 command line option, 24
`color_series()` (in module `euporie.style`), 158
`ColorQueryMixin` (class in `euporie.terminal`), 160, 162
`Command` (class in `euporie.commands`), 91
`Command` (class in `euporie.commands.base`), 64
command line option
 -V, 17
 --autocomplete, 18
 --autoformat, 18
 --autoinspect, 18
 --autosuggest, 18
 --background-character, 18
 --background-color, 18
 --background-pattern, 18
 --bg-char, 18
 --bg-color, 18
 --bg-pattern, 18
 --color-scheme, 18
 --debug, 17
 --dump, 17
 --dump-file, 17
 --edit-mode, 18
 --expand, 18
 --format-black, 18
 --format-isort, 18
 --format-ssort, 18
 --help, 17
 --line-numbers, 18
 --log-file, 17
 --max-notebook-width, 18
 --no-autocomplete, 18
 --no-autoformat, 18
 --no-autoinspect, 18
 --no-autosuggest, 18
 --no-debug, 17
 --no-dump, 17
 --no-expand, 18
 --no-format-black, 18
 --no-format-isort, 18
 --no-format-ssort, 18
 --no-line-numbers, 18
 --no-page, 17
 --no-run, 17
 --no-show-cell-borders, 18
 --no-show-status-bar, 18
 --no-tmux-graphics, 18
 --page, 17
 --run, 17
 --run-after-external-edit, 18
 --show-cell-borders, 18
 --show-status-bar, 18
 --syntax-theme, 19
 --tab-size, 18
 --terminal-polling-interval, 18
 --tmux-graphics, 18
 --version, 17
-h, 17
<Path, 17
autocomplete, 23
autoformat, 23
autoinspect, 24
autosuggest, 23
background_character, 25
background_color, 25

- background_pattern, 25
- color_scheme, 24
- debug, 20
- dump, 21
- dump_file, 21
- edit_mode, 22
- expand, 24
- files, 26
- format_black, 22
- format_isort, 23
- format_ssort, 23
- line_numbers, 25
- log_file, 20
- max_notebook_width, 24
- page, 21
- run, 21
- run_after_external_edit, 22
- show_cell_borders, 25
- show_status_bar, 24
- syntax_theme, 26
- tab_size, 22
- terminal_polling_interval, 22
- tmux_graphics, 21
- version, 20
- command_palette (*euporie.app.dump.DumpApp* attribute), 44
- command_palette (*euporie.app.tui.TuiApp* attribute), 49
- CommandMenuControl (class in *euporie.palette*), 153
- CommandPalette (class in *euporie.palette*), 153
- commands_exist() (in module *euporie.convert.util*), 110
- complete() (*euporie.kernel.NotebookKernel* method), 112
- complete_() (*euporie.kernel.NotebookKernel* method), 113
- ConditionalAutoSuggestAsync (class in *euporie.suggest*), 159
- conf_file_name (*euporie.config.Config* attribute), 94
- Config (class in *euporie.config*), 93
- container (*euporie.log.LogView* attribute), 121
- container (*euporie.notebook.Notebook* attribute), 145
- container (*euporie.tab.Tab* attribute), 159
- container_statuses (*euporie.app.dump.DumpApp* attribute), 44
- container_statuses (*euporie.app.tui.TuiApp* attribute), 49
- content (*euporie.output.container.GraphicWindow* attribute), 148
- context (*euporie.app.dump.DumpApp* attribute), 44
- context (*euporie.app.tui.TuiApp* attribute), 49
- continue_() (*euporie.markdown.blocks.math.MathBlock* static method), 125
- continue_() (*euporie.markdown.blocks.MathBlock* static method), 127
- continue_() (*euporie.markdown.blocks.Table* static method), 127
- continue_() (*euporie.markdown.blocks.tables.Table* static method), 126
- convert() (in module *euporie.convert.base*), 97
- Convertor (class in *euporie.convert.base*), 97
- copy() (*euporie.notebook.Notebook* method), 145
- copy_cells() (in module *euporie.commands.notebook*), 82, 85
- copy_selection() (in module *euporie.commands.buffer*), 66, 71
- count() (*euporie.convert.base.Convertor* method), 97
- cpr_not_supported_callback() (*euporie.app.base.EuporieApp* method), 38
- cpr_not_supported_callback() (*euporie.app.dump.DumpApp* method), 44
- cpr_not_supported_callback() (*euporie.app.tui.TuiApp* method), 49
- create() (*euporie.markdown.LatexBlock* class method), 136
- create() (*euporie.markdown.LatexElement* class method), 137
- create() (*euporie.markdown.LatexInline* class method), 138
- create() (*euporie.markdown.rich.LatexBlock* class method), 131
- create() (*euporie.markdown.rich.LatexElement* class method), 132
- create() (*euporie.markdown.rich.LatexInline* class method), 133
- create() (*euporie.markdown.rich.Table* class method), 134
- create() (*euporie.markdown.Table* class method), 140
- create_background_task() (*euporie.app.base.EuporieApp* method), 39
- create_background_task() (*euporie.app.dump.DumpApp* method), 44
- create_background_task() (*euporie.app.tui.TuiApp* method), 49
- create_content() (*euporie.output.control.AnsiControl* method), 151
- create_content() (*euporie.palette.CommandMenuControl* method), 153
- create_content() (*euporie.scroll.ScrollbarControl* method), 155
- create_margin() (*euporie.margins.ScrollbarMargin* method), 124
- create_merged_style() (*euporie.app.base.EuporieApp* method), 39
- create_merged_style() (*euporie.app.dump.DumpApp* method), 44
- create_merged_style() (*euporie.app.tui.TuiApp* method), 49

method), 49
createLock() (euporie.log.QueueHandler method), 122
CROSS (euporie.box.RoundBorder attribute), 56
CROSS (euporie.box.SquareBorder attribute), 57
current_buffer (euporie.app.base.EuporieApp property), 39
current_buffer (euporie.app.dump.DumpApp property), 44
current_buffer (euporie.app.tui.TuiApp property), 49
current_search_state (euporie.app.base.EuporieApp property), 39
current_search_state (euporie.app.dump.DumpApp property), 44
current_search_state (euporie.app.tui.TuiApp property), 50
cursor_position (euporie.output.control.AnsiControl property), 151
cut() (euporie.notebook.Notebook method), 145
cut_cells() (in module euporie.commands.notebook), 82, 85
cut_line() (in module euporie.commands.buffer), 66, 71
cut_selection() (in module euporie.commands.buffer), 66, 71

D

data (euporie.output.container.CellOutput property), 148
debug
 command line option, 20
default (euporie.terminal.BackgroundColor attribute), 162
default (euporie.terminal.DepthOfColor attribute), 162
default (euporie.terminal.ForegroundColor attribute), 163
default (euporie.terminal.ItermGraphicsStatus attribute), 163
default (euporie.terminal.KittyGraphicsStatus attribute), 163
default (euporie.terminal.PixelDimensions attribute), 164
default (euporie.terminal.SixelGraphicsStatus attribute), 164
default (euporie.terminal.TerminalQuery attribute), 165
default() (euporie.config.JSONEncoderPlus method), 94
defaults (euporie.config.Config attribute), 94
delete() (euporie.notebook.Notebook method), 145
delete_cells() (in module euporie.commands.notebook), 82, 85
delete_selection() (in module euporie.commands.buffer), 67, 71

dent_buffer() (in module euporie.commands.buffer), 67, 71
DepthOfColor (class in euporie.terminal), 161, 162
details (euporie.keys.KeyBindingsInfo attribute), 119
dialog() (euporie.app.tui.TuiApp method), 50
dict_bindings() (in module euporie.key_binding.util), 118
disabled (euporie.menu.item.MenuItem property), 142
disabled (euporie.menu.MenuItem property), 143
document (euporie.cell.CellInputTextArea property), 60
document (euporie.cell.CellStdinTextArea property), 61
document (euporie.text.FormattedTextArea property), 166
dumb_console (euporie.markdown.rich.Table attribute), 134
dumb_console (euporie.markdown.Table attribute), 140
dump
 command line option, 21
dump_file
 command line option, 21
DumpApp (class in euporie.app.dump), 43
DumpKernelNotebook (class in euporie.notebook), 145
DumpNotebook (class in euporie.notebook), 145
duplicate_line() (in module euporie.commands.buffer), 67, 71
duplicate_selection() (in module euporie.commands.buffer), 67, 71

E

edit_in_editor() (euporie.cell.InteractiveCell method), 61
edit_in_external_editor() (in module euporie.commands.cell), 73
edit_mode
 command line option, 22
eighths (euporie.margins.ScrollbarMargin attribute), 124
elements (euporie.markdown.Markdown attribute), 139
elements (euporie.markdown.rich.Markdown attribute), 134
emit() (euporie.log.QueueHandler method), 122
encode() (euporie.config.JSONEncoderPlus method), 94
end_macro() (euporie.key_binding.micro_state.MicroState method), 117
end_macro() (in module euporie.commands.buffer), 67, 71
enter_cell_edit_mode() (in module euporie.commands.cell), 73
enter_edit_mode() (euporie.cell.InteractiveCell method), 61
environment variable
 EUPORIE_AUTOCOMPLETE, 23
 EUPORIE_AUTOFORMAT, 23

- EUPORIE_AUTOINSPECT, 24
- EUPORIE_AUTOSUGGEST, 24
- EUPORIE_BACKGROUND_CHARACTER, 25
- EUPORIE_BACKGROUND_COLOR, 25
- EUPORIE_BACKGROUND_PATTERN, 25
- EUPORIE_COLOR_SCHEME, 24
- EUPORIE_DEBUG, 20
- EUPORIE_DUMP, 21
- EUPORIE_DUMP_FILE, 21
- EUPORIE_EDIT_MODE, 22
- EUPORIE_EXPAND, 24
- EUPORIE_FILES, 26
- EUPORIE_FORMAT_BLACK, 23
- EUPORIE_FORMAT_ISORT, 23
- EUPORIE_FORMAT_SSort, 23
- EUPORIE_LINE_NUMBERS, 26
- EUPORIE_LOG_FILE, 20
- EUPORIE_MAX_NOTEBOOK_WIDTH, 24
- EUPORIE_PAGE, 21
- EUPORIE_RUN, 21
- EUPORIE_RUN_AFTER_EXTERNAL_EDIT, 22
- EUPORIE_SHOW_CELL_BORDERS, 25
- EUPORIE_SHOW_STATUS_BAR, 24
- EUPORIE_SYNTAX_THEME, 26
- EUPORIE_TAB_SIZE, 22
- EUPORIE_TERMINAL_POLLING_INTERVAL, 22
- EUPORIE_TMUX_GRAPHICS, 21
- NO_COLOR, 34
- euporie
 - module, 36
- euporie.app
 - module, 37
- euporie.app.base
 - module, 38
- euporie.app.current
 - module, 42
- euporie.app.dump
 - module, 43
- euporie.app.tui
 - module, 48
- euporie.box
 - module, 54
- euporie.cell
 - module, 57
- euporie.commands
 - module, 63
- euporie.commands.base
 - module, 63
- euporie.commands.buffer
 - module, 65
- euporie.commands.cell
 - module, 73
- euporie.commands.cell_output
 - module, 74
- euporie.commands.completions
 - module, 75
- euporie.commands.config
 - module, 76
- euporie.commands.format
 - module, 79
- euporie.commands.notebook
 - module, 80
- euporie.commands.pager
 - module, 87
- euporie.commands.registry
 - module, 87
- euporie.commands.search
 - module, 88
- euporie.commands.suggestions
 - module, 88
- euporie.commands.tui
 - module, 89
- euporie.completion
 - module, 92
- euporie.config
 - module, 93
- euporie.containers
 - module, 95
- euporie.convert
 - module, 96
- euporie.convert.base
 - module, 96
- euporie.convert.formats
 - module, 98
- euporie.convert.formats.ansi
 - module, 98
- euporie.convert.formats.base64
 - module, 104
- euporie.convert.formats.common
 - module, 104
- euporie.convert.formats.jpeg
 - module, 105
- euporie.convert.formats.markdown
 - module, 105
- euporie.convert.formats.pdf
 - module, 106
- euporie.convert.formats.pil
 - module, 106
- euporie.convert.formats.png
 - module, 107
- euporie.convert.formats.rich
 - module, 108
- euporie.convert.formats.sixel
 - module, 108
- euporie.convert.util
 - module, 109
- euporie.filters
 - module, 110

euporie.format
 module, 110

euporie.kernel
 module, 111

euporie.key_binding
 module, 115

euporie.key_binding.bindings
 module, 116

euporie.key_binding.bindings.commands
 module, 116

euporie.key_binding.bindings.micro
 module, 116

euporie.key_binding.micro_state
 module, 117

euporie.key_binding.util
 module, 118

euporie.keys
 module, 118

euporie.log
 module, 120

euporie.margins
 module, 123

euporie.markdown
 module, 124

euporie.markdown.blocks
 module, 125

euporie.markdown.blocks.math
 module, 125

euporie.markdown.blocks.tables
 module, 125

euporie.markdown.inlines
 module, 127

euporie.markdown.parser
 module, 129

euporie.markdown.rich
 module, 131

euporie.menu
 module, 141

euporie.menu.bar
 module, 141

euporie.menu.contents
 module, 142

euporie.menu.item
 module, 142

euporie.notebook
 module, 144

euporie.output
 module, 146

euporie.output.container
 module, 147

euporie.output.control
 module, 150

euporie.palette
 module, 152

euporie.scroll
 module, 154

euporie.style
 module, 158

euporie.suggest
 module, 158

euporie.tab
 module, 159

euporie.terminal
 module, 160

euporie.text
 module, 166

EUPORIE_AUTOCOMPLETE, 23

EUPORIE_AUTOFORMAT, 23

EUPORIE_AUTOINSPECT, 24

EUPORIE_AUTOSUGGEST, 24

EUPORIE_BACKGROUND_CHARACTER, 25

EUPORIE_BACKGROUND_COLOR, 25

EUPORIE_BACKGROUND_PATTERN, 25

EUPORIE_COLOR_SCHEME, 24

EUPORIE_DEBUG, 20

EUPORIE_DUMP, 21

EUPORIE_DUMP_FILE, 21

EUPORIE_EDIT_MODE, 22

EUPORIE_EXPAND, 24

EUPORIE_FILES, 26

EUPORIE_FORMAT_BLACK, 23

EUPORIE_FORMAT_ISORT, 23

EUPORIE_FORMAT_SSHORT, 23

EUPORIE_LINE_NUMBERS, 26

EUPORIE_LOG_FILE, 20

EUPORIE_MAX_NOTEBOOK_WIDTH, 24

EUPORIE_PAGE, 21

EUPORIE_RUN, 21

EUPORIE_RUN_AFTER_EXTERNAL_EDIT, 22

EUPORIE_SHOW_CELL_BORDERS, 25

EUPORIE_SHOW_STATUS_BAR, 24

EUPORIE_SYNTAX_THEME, 26

EUPORIE_TAB_SIZE, 22

EUPORIE_TERMINAL_POLLING_INTERVAL, 22

EUPORIE_TMUX_GRAPHICS, 21

EuporieApp (*class in euporie.app.base*), 38

execution_count (*euporie.cell.Cell property*), 59

execution_count (*euporie.cell.InteractiveCell property*), 61

exit() (*euporie.app.base.EuporieApp method*), 39

exit() (*euporie.app.dump.DumpApp method*), 44

exit() (*euporie.app.tui.TuiApp method*), 50

exit_edit_mode() (*euporie.cell.Cell method*), 59

exit_edit_mode() (*euporie.cell.InteractiveCell method*), 61

exit_edit_mode() (*in module euporie.commands.cell*), 73

expand

- command line option, 24
- `extend_cell_selection_down()` (in module *euporie.commands.notebook*), 82, 86
- `extend_cell_selection_up()` (in module *euporie.commands.notebook*), 82, 86
- `extend_selection()` (in module *euporie.commands.buffer*), 67, 71
- F**
 - `feed()` (*euporie.terminal.Vt100Parser* method), 165
 - `feed_and_flush()` (*euporie.terminal.Vt100Parser* method), 165
 - `fenced_code_block()` (*euporie.markdown.blocks.BlockStarts* static method), 126
 - files
 - command line option, 26
 - `fill_sugestion()` (in module *euporie.commands.suggestions*), 88
 - `filter()` (*euporie.log.QueueHandler* method), 122
 - `finalize()` (*euporie.markdown.blocks.math.MathBlock* static method), 125
 - `finalize()` (*euporie.markdown.blocks.MathBlock* static method), 127
 - `finalize()` (*euporie.markdown.blocks.Table* static method), 127
 - `finalize()` (*euporie.markdown.blocks.tables.Table* static method), 126
 - `finalize()` (*euporie.markdown.Parser* method), 140
 - `finalize()` (*euporie.markdown.parser.Parser* method), 130
 - `find_next_nonspace()` (*euporie.markdown.Parser* method), 140
 - `find_next_nonspace()` (*euporie.markdown.parser.Parser* method), 130
 - `find_route()` (in module *euporie.convert.base*), 97
 - floats (*euporie.app.dump.DumpApp* attribute), 45
 - floats (*euporie.app.tui.TuiApp* attribute), 50
 - floats (*euporie.menu.bar.MenuContainer* property), 141
 - floats (*euporie.menu.MenuContainer* property), 143
 - `flush()` (*euporie.log.QueueHandler* method), 122
 - `flush()` (*euporie.terminal.Vt100Parser* method), 165
 - `focus()` (*euporie.log.LogView* method), 121
 - `focus()` (*euporie.notebook.Notebook* method), 145
 - `focus()` (*euporie.tab.Tab* method), 159
 - `focus_next()` (in module *euporie.commands.tui*), 89, 91
 - `focus_previous()` (in module *euporie.commands.tui*), 89, 91
 - focused (*euporie.cell.Cell* property), 59
 - focused (*euporie.cell.InteractiveCell* property), 61
 - ForegroundColor (class in *euporie.terminal*), 161, 162
 - `format()` (*euporie.log.QueueHandler* method), 122
 - `format()` (*euporie.text.ANSI* method), 166
 - `format_black`
 - command line option, 22
 - `format_black()` (in module *euporie.commands.config*), 77, 78
 - `format_black()` (in module *euporie.format*), 111
 - `format_code()` (in module *euporie.format*), 111
 - `format_command_attrs()` (in module *euporie.commands.format*), 79, 80
 - `format_isort`
 - command line option, 23
 - `format_isort()` (in module *euporie.commands.config*), 77, 78
 - `format_isort()` (in module *euporie.format*), 111
 - `format_keys()` (in module *euporie.key_binding.util*), 118
 - `format_ssort`
 - command line option, 23
 - `format_ssort()` (in module *euporie.commands.config*), 77, 79
 - `format_ssort()` (in module *euporie.format*), 111
 - `format_status()` (*euporie.app.tui.TuiApp* method), 50
 - `format_title()` (*euporie.app.tui.TuiApp* method), 50
 - `format_usage()` (*euporie.config.BooleanOptionalAction* method), 93
 - formatted_text (*euporie.menu.item.MenuItem* property), 142
 - formatted_text (*euporie.menu.MenuItem* property), 144
 - formatted_text (*euporie.text.FormattedTextArea* property), 166
 - FormattedTextArea (class in *euporie.text*), 166
 - FormatTextProcessor (class in *euporie.text*), 166
 - `from_command()` (*euporie.menu.item.MenuItem* class method), 143
 - `from_command()` (*euporie.menu.MenuItem* class method), 144
 - full_screen (*euporie.app.dump.DumpApp* attribute), 45
 - full_screen (*euporie.app.tui.TuiApp* attribute), 50
 - func (*euporie.convert.base.Convertor* attribute), 97
 - future (*euporie.app.dump.DumpApp* attribute), 45
 - future (*euporie.app.tui.TuiApp* attribute), 50
- G**
 - `get()` (*euporie.config.Config* method), 94
 - `get()` (in module *euporie.commands*), 92
 - `get()` (in module *euporie.commands.base*), 64, 65
 - `get()` (in module *euporie.commands.registry*), 88
 - `get_base_app()` (in module *euporie.app.current*), 43
 - `get_bindings_for_keys()` (*euporie.keys.KeyBindingsInfo* method), 119
 - `get_bindings_starting_with_keys()` (*euporie.keys.KeyBindingsInfo* method), 119

`get_cell_by_id()` (*euporie.notebook.Notebook* method), 145
`get_cell_id()` (in module *euporie.cell*), 58, 62
`get_child()` (*euporie.scroll.ScrollingContainer* method), 156
`get_child_render_info()` (*euporie.scroll.ScrollingContainer* method), 156
`get_children()` (*euporie.box.BorderLine* method), 55
`get_children()` (*euporie.box.Pattern* method), 56
`get_children()` (*euporie.containers.PrintingContainer* method), 95
`get_children()` (*euporie.output.container.GraphicWindow* method), 148
`get_children()` (*euporie.output.container.OutputWindow* method), 149
`get_children()` (*euporie.scroll.ScrollingContainer* method), 156
`get_completions()` (*euporie.completion.KernelCompleter* method), 92
`get_completions_async()` (*euporie.completion.KernelCompleter* method), 92
`get_dims()` (in module *euporie.output.container*), 147, 149
`get_dump_app()` (in module *euporie.app.current*), 43
`get_edit_mode()` (*euporie.app.base.EuporieApp* method), 39
`get_edit_mode()` (*euporie.app.dump.DumpApp* method), 45
`get_edit_mode()` (*euporie.app.tui.TuiApp* method), 50
`get_input()` (*euporie.cell.InteractiveCell* method), 61
`get_invalidate_events()` (*euporie.output.control.AnsiControl* method), 152
`get_invalidate_events()` (*euporie.palette.CommandMenuControl* method), 153
`get_invalidate_events()` (*euporie.scroll.ScrollbarControl* method), 155
`get_key_bindings()` (*euporie.box.BorderLine* method), 55
`get_key_bindings()` (*euporie.box.Pattern* method), 56
`get_key_bindings()` (*euporie.containers.PrintingContainer* method), 95
`get_key_bindings()` (*euporie.output.container.GraphicWindow* method), 148
`get_key_bindings()` (*euporie.output.container.OutputWindow* method), 149
`get_key_bindings()` (*euporie.output.control.AnsiControl* method), 152
`get_key_bindings()` (*euporie.palette.CommandMenuControl* method), 153
`get_key_bindings()` (*euporie.scroll.ScrollbarControl* method), 155
`get_key_bindings()` (*euporie.scroll.ScrollingContainer* method), 156
`get_markdown_width()` (*euporie.markdown.rich.Table* method), 134
`get_markdown_width()` (*euporie.markdown.Table* method), 140
`get_name()` (*euporie.log.QueueHandler* method), 122
`get_processor()` (*euporie.text.FormattedTextArea* method), 166
`get_rendered_lines()` (*euporie.output.control.AnsiControl* method), 152
`get_suggestion()` (*euporie.suggest.KernelAutoSuggest* method), 159
`get_suggestion_async()` (*euporie.suggest.KernelAutoSuggest* method), 159
`get_tui_app()` (in module *euporie.app.current*), 43
`get_used_style_strings()` (*euporie.app.base.EuporieApp* method), 39
`get_used_style_strings()` (*euporie.app.dump.DumpApp* method), 45
`get_used_style_strings()` (*euporie.app.tui.TuiApp* method), 50
`get_width()` (*euporie.margins.ScrollbarMargin* method), 124
`go_to_end_of_cell_output()` (in module *euporie.commands.cell_output*), 74, 75
`go_to_end_of_line()` (in module *euporie.commands.buffer*), 67, 71
`go_to_end_of_paragraph()` (in module *euporie.commands.buffer*), 67, 71
`go_to_start_of_cell_output()` (in module *euporie.commands.cell_output*), 74, 75
`go_to_start_of_line()` (in module *euporie.commands.buffer*), 68, 71
`go_to_start_of_paragraph()` (in module *euporie.commands.buffer*), 68, 71
`GraphicWindow` (class in *euporie.output.container*), 148

H

`handle()` (*euporie.log.QueueHandler* method), 122
`handleDelim()` (*euporie.markdown.InlineParser* method), 135

- [handleDelim\(\)](#) (*euporie.markdown.inlines.InlineParser* method), 128
[handleError\(\)](#) (*euporie.log.QueueHandler* method), 122
[has_toggles](#) (*euporie.menu.item.MenuItem* property), 143
[has_toggles](#) (*euporie.menu.MenuItem* property), 144
[have_modules\(\)](#) (in module *euporie.convert.util*), 110
[help_about\(\)](#) (*euporie.app.tui.TuiApp* method), 50
[help_keys\(\)](#) (*euporie.app.tui.TuiApp* method), 50
[help_logs\(\)](#) (*euporie.app.tui.TuiApp* method), 50
[hide\(\)](#) (*euporie.output.control.AnsiControl* method), 152
[hide\(\)](#) (*euporie.palette.CommandPalette* method), 153
[hide_pager\(\)](#) (*euporie.notebook.Notebook* method), 145
[history\(\)](#) (*euporie.kernel.NotebookKernel* method), 113
[history_\(\)](#) (*euporie.kernel.NotebookKernel* method), 113
[hook\(\)](#) (*euporie.log.QueueHandler* class method), 122
[hook_id](#) (*euporie.log.QueueHandler* attribute), 122
[hooks](#) (*euporie.log.QueueHandler* attribute), 122
[HORIZONTAL](#) (*euporie.box.RoundBorder* attribute), 56
[HORIZONTAL](#) (*euporie.box.SquareBorder* attribute), 57
[html_block\(\)](#) (*euporie.markdown.blocks.BlockStarts* static method), 126
[html_to_ansi_elinks\(\)](#) (in module *euporie.convert.formats.ansi*), 99, 102
[html_to_ansi_links\(\)](#) (in module *euporie.convert.formats.ansi*), 99, 102
[html_to_ansi_lynx\(\)](#) (in module *euporie.convert.formats.ansi*), 99, 102
[html_to_ansi_py_htmlparser\(\)](#) (in module *euporie.convert.formats.ansi*), 99, 102
[html_to_ansi_w3m\(\)](#) (in module *euporie.convert.formats.ansi*), 99, 102
[html_to_markdown_py_mtable\(\)](#) (in module *euporie.convert.formats.markdown*), 106

I
[id](#) (*euporie.cell.Cell* property), 59
[id](#) (*euporie.cell.InteractiveCell* property), 61
[id](#) (*euporie.kernel.NotebookKernel* property), 113
[if_no_repeat\(\)](#) (in module *euporie.commands.buffer*), 68, 71
[image_to_ansi_catimg\(\)](#) (in module *euporie.convert.formats.ansi*), 100, 102
[image_to_ansi_icat\(\)](#) (in module *euporie.convert.formats.ansi*), 100, 102
[image_to_ansi_jp2a\(\)](#) (in module *euporie.convert.formats.ansi*), 100, 102
[image_to_ansi_timg\(\)](#) (in module *euporie.convert.formats.ansi*), 100, 102
[image_to_ansi_tiv\(\)](#) (in module *euporie.convert.formats.ansi*), 100, 103
[image_to_ansi_viu\(\)](#) (in module *euporie.convert.formats.ansi*), 100, 103
[imagemagick_convert\(\)](#) (in module *euporie.convert.formats.common*), 105
[incorporate_line\(\)](#) (*euporie.markdown.Parser* method), 140
[incorporate_line\(\)](#) (*euporie.markdown.parser.Parser* method), 130
[indent_lines\(\)](#) (in module *euporie.commands.buffer*), 68, 71
[indented_code_block\(\)](#) (*euporie.markdown.blocks.BlockStarts* static method), 127
[index](#) (*euporie.palette.CommandPalette* attribute), 153
[index\(\)](#) (*euporie.convert.base.Convertor* method), 97
[info\(\)](#) (*euporie.kernel.NotebookKernel* method), 113
[info_\(\)](#) (*euporie.kernel.NotebookKernel* method), 113
[InlineParser](#) (class in *euporie.markdown*), 135
[InlineParser](#) (class in *euporie.markdown.inlines*), 128
[inlines](#) (*euporie.markdown.Markdown* attribute), 139
[inlines](#) (*euporie.markdown.rich.Markdown* attribute), 134
[INNER_VERTICAL](#) (*euporie.box.RoundBorder* attribute), 57
[INNER_VERTICAL](#) (*euporie.box.SquareBorder* attribute), 57
[input](#) (*euporie.cell.Cell* property), 59
[input](#) (*euporie.cell.InteractiveCell* property), 61
[input](#) (*euporie.terminal.TerminalInfo* attribute), 164
[InputMode](#) (class in *euporie.key_binding.micro_state*), 117
[INSERT](#) (*euporie.key_binding.micro_state.InputMode* attribute), 117
[inspect\(\)](#) (*euporie.cell.Cell* method), 59
[inspect\(\)](#) (*euporie.cell.InteractiveCell* method), 62
[inspect\(\)](#) (*euporie.kernel.NotebookKernel* method), 113
[inspect_\(\)](#) (*euporie.kernel.NotebookKernel* method), 113
[InteractiveCell](#) (class in *euporie.cell*), 59, 61
[interrupt\(\)](#) (*euporie.kernel.NotebookKernel* method), 113
[interrupt_kernel\(\)](#) (in module *euporie.commands.notebook*), 82, 86
[invalidate\(\)](#) (*euporie.app.base.EuporieApp* method), 39
[invalidate\(\)](#) (*euporie.app.dump.DumpApp* method), 45
[invalidate\(\)](#) (*euporie.app.tui.TuiApp* method), 50
[invalidated](#) (*euporie.app.base.EuporieApp* property), 39

- `invalidated` (*euporie.app.dump.DumpApp* property), 45
- `invalidated` (*euporie.app.tui.TuiApp* property), 51
- `is_done` (*euporie.app.base.EuporieApp* property), 39
- `is_done` (*euporie.app.dump.DumpApp* property), 45
- `is_done` (*euporie.app.tui.TuiApp* property), 51
- `is_focusable()` (*euporie.output.control.AnsiControl* method), 152
- `is_focusable()` (*euporie.palette.CommandMenuControl* method), 153
- `is_focusable()` (*euporie.scroll.ScrollbarControl* method), 155
- `is_modal()` (*euporie.box.BorderLine* method), 55
- `is_modal()` (*euporie.box.Pattern* method), 56
- `is_modal()` (*euporie.cell.CellStdinTextArea* method), 61
- `is_modal()` (*euporie.containers.PrintingContainer* method), 95
- `is_modal()` (*euporie.output.container.GraphicWindow* method), 149
- `is_modal()` (*euporie.output.container.OutputWindow* method), 149
- `is_modal()` (*euporie.scroll.ScrollingContainer* method), 156
- `is_recording` (*euporie.key_binding.micro_state.MicroState* property), 117
- `is_running` (*euporie.app.base.EuporieApp* property), 39
- `is_running` (*euporie.app.dump.DumpApp* property), 45
- `is_running` (*euporie.app.tui.TuiApp* property), 51
- `item_separator` (*euporie.config.JSONEncoderPlus* attribute), 95
- `iterencode()` (*euporie.config.JSONEncoderPlus* method), 95
- `ItermGraphicControl` (class in *euporie.output.control*), 151
- `ItermGraphicsStatus` (class in *euporie.terminal*), 161, 163
- J**
- `JSONEncoderPlus` (class in *euporie.config*), 93, 94
- `justify` (*euporie.markdown.LatexBlock* attribute), 137
- `justify` (*euporie.markdown.rich.LatexBlock* attribute), 132
- `justify` (*euporie.markdown.rich.Markdown* attribute), 134
- K**
- `KernelAutoSuggest` (class in *euporie.suggest*), 159
- `KernelCompleter` (class in *euporie.completion*), 92
- `KernelNotebook` (class in *euporie.notebook*), 145
- `key_bindings` (*euporie.commands.base.Command* property), 64
- `key_bindings` (*euporie.commands.Command* property), 92
- `key_handler` (*euporie.commands.base.Command* property), 64
- `key_handler` (*euporie.commands.Command* property), 92
- `key_processor` (*euporie.app.base.EuporieApp* attribute), 39
- `key_processor` (*euporie.app.dump.DumpApp* attribute), 45
- `key_processor` (*euporie.app.tui.TuiApp* attribute), 51
- `key_separator` (*euporie.config.JSONEncoderPlus* attribute), 95
- `KeyBindingsInfo` (class in *euporie.keys*), 118
- `keyboard_shortcuts()` (in module *euporie.commands.tui*), 90, 91
- `KittyGraphicControl` (class in *euporie.output.control*), 151
- `KittyGraphicsStatus` (class in *euporie.terminal*), 161, 163
- `known_sizes` (*euporie.scroll.ScrollingContainer* property), 156
- L**
- `lang_file_ext()` (*euporie.notebook.Notebook* method), 145
- `language` (*euporie.cell.Cell* property), 59
- `language` (*euporie.cell.InteractiveCell* property), 62
- `latex_to_ansi_py_flatlatex()` (in module *euporie.convert.formats.ansi*), 101, 103
- `latex_to_ansi_py_pylatexenc()` (in module *euporie.convert.formats.ansi*), 101, 103
- `latex_to_ansi_py_sympy()` (in module *euporie.convert.formats.ansi*), 101, 103
- `latex_to_png_py_ipython()` (in module *euporie.convert.formats.png*), 107
- `LatexBlock` (class in *euporie.markdown*), 136
- `LatexBlock` (class in *euporie.markdown.rich*), 131
- `LatexElement` (class in *euporie.markdown*), 137
- `LatexElement` (class in *euporie.markdown.rich*), 131, 132
- `LatexInline` (class in *euporie.markdown*), 138
- `LatexInline` (class in *euporie.markdown.rich*), 131, 133
- `launch()` (*euporie.app.base.EuporieApp* class method), 39
- `launch()` (*euporie.app.dump.DumpApp* class method), 45
- `launch()` (*euporie.app.tui.TuiApp* class method), 51
- `line_numbers`
 - command line option, 25
- `list_item()` (*euporie.markdown.blocks.BlockStarts* static method), 127
- `load()` (*euporie.config.Config* method), 94
- `load_args()` (*euporie.config.Config* method), 94
- `load_clipboard()` (*euporie.app.tui.TuiApp* method), 51

[load_command_bindings\(\)](#) (in module *euporie.key_binding.bindings.commands*), 116
[load_container\(\)](#) (*euporie.app.base.EuporieApp* method), 40
[load_container\(\)](#) (*euporie.app.dump.DumpApp* method), 45
[load_container\(\)](#) (*euporie.app.tui.TuiApp* method), 51
[load_env\(\)](#) (*euporie.config.Config* method), 94
[load_key_bindings\(\)](#) (*euporie.app.base.EuporieApp* method), 40
[load_key_bindings\(\)](#) (*euporie.app.dump.DumpApp* method), 45
[load_key_bindings\(\)](#) (*euporie.app.tui.TuiApp* method), 51
[load_menu_items\(\)](#) (in module *euporie.menu.contents*), 142
[load_micro_bindings\(\)](#) (in module *euporie.key_binding.bindings.micro*), 117
[load_output\(\)](#) (*euporie.app.base.EuporieApp* method), 40
[load_output\(\)](#) (*euporie.app.dump.DumpApp* method), 45
[load_output\(\)](#) (*euporie.app.tui.TuiApp* method), 51
[load_parser\(\)](#) (*euporie.config.Config* method), 94
[load_tabs\(\)](#) (*euporie.app.dump.DumpApp* method), 45
[load_user\(\)](#) (*euporie.config.Config* method), 94
[log_file](#)
 command line option, 20
[LogView](#) (class in *euporie.log*), 121
[loop](#) (*euporie.app.dump.DumpApp* attribute), 45
[loop](#) (*euporie.app.tui.TuiApp* attribute), 51

M

[Markdown](#) (class in *euporie.markdown*), 139
[Markdown](#) (class in *euporie.markdown.rich*), 131, 134
[markdown_to_rich_py\(\)](#) (in module *euporie.convert.formats.ansi*), 101, 103
[markdown_to_rich_py\(\)](#) (in module *euporie.convert.formats.rich*), 108
[match\(\)](#) (*euporie.markdown.InlineParser* method), 135
[match\(\)](#) (*euporie.markdown.inlines.InlineParser* method), 128
[matches](#) (*euporie.palette.CommandPalette* attribute), 154
[math_block\(\)](#) (*euporie.markdown.blocks.BlockStarts* static method), 127
[MathBlock](#) (class in *euporie.markdown.blocks*), 127
[MathBlock](#) (class in *euporie.markdown.blocks.math*), 125
[max_cols](#) (*euporie.output.control.AnsiControl* property), 152
[max_notebook_width](#)
 command line option, 24
[menu](#) (*euporie.commands.base.Command* property), 65
[menu](#) (*euporie.commands.Command* property), 92
[menu_container](#) (*euporie.app.tui.TuiApp* attribute), 51
[menu_handler](#) (*euporie.commands.base.Command* property), 65
[menu_handler](#) (*euporie.commands.Command* property), 92
[MenuContainer](#) (class in *euporie.menu*), 143
[MenuContainer](#) (class in *euporie.menu.bar*), 141
[MenuItem](#) (class in *euporie.menu*), 143
[MenuItem](#) (class in *euporie.menu.item*), 142
[merge\(\)](#) (*euporie.notebook.Notebook* method), 145
[merge_cells\(\)](#) (in module *euporie.commands.notebook*), 82, 86
[METHODS](#) (*euporie.markdown.blocks.BlockStarts* attribute), 126
[MicroState](#) (class in *euporie.key_binding.micro_state*), 117
[missing](#) (*euporie.kernel.NotebookKernel* property), 113
[module](#)
 euporie, 36
 euporie.app, 37
 euporie.app.base, 38
 euporie.app.current, 42
 euporie.app.dump, 43
 euporie.app.tui, 48
 euporie.box, 54
 euporie.cell, 57
 euporie.commands, 63
 euporie.commands.base, 63
 euporie.commands.buffer, 65
 euporie.commands.cell, 73
 euporie.commands.cell_output, 74
 euporie.commands.completions, 75
 euporie.commands.config, 76
 euporie.commands.format, 79
 euporie.commands.notebook, 80
 euporie.commands.pager, 87
 euporie.commands.registry, 87
 euporie.commands.search, 88
 euporie.commands.suggestions, 88
 euporie.commands.tui, 89
 euporie.completion, 92
 euporie.config, 93
 euporie.containers, 95
 euporie.convert, 96
 euporie.convert.base, 96
 euporie.convert.formats, 98
 euporie.convert.formats.ansi, 98
 euporie.convert.formats.base64, 104
 euporie.convert.formats.common, 104
 euporie.convert.formats.jpeg, 105
 euporie.convert.formats.markdown, 105

- euporie.convert.formats.pdf, 106
 - euporie.convert.formats.pil, 106
 - euporie.convert.formats.png, 107
 - euporie.convert.formats.rich, 108
 - euporie.convert.formats.sixel, 108
 - euporie.convert.util, 109
 - euporie.filters, 110
 - euporie.format, 110
 - euporie.kernel, 111
 - euporie.key_binding, 115
 - euporie.key_binding.bindings, 116
 - euporie.key_binding.bindings.commands, 116
 - euporie.key_binding.bindings.micro, 116
 - euporie.key_binding.micro_state, 117
 - euporie.key_binding.util, 118
 - euporie.keys, 118
 - euporie.log, 120
 - euporie.margins, 123
 - euporie.markdown, 124
 - euporie.markdown.blocks, 125
 - euporie.markdown.blocks.math, 125
 - euporie.markdown.blocks.tables, 125
 - euporie.markdown.inlines, 127
 - euporie.markdown.parser, 129
 - euporie.markdown.rich, 131
 - euporie.menu, 141
 - euporie.menu.bar, 141
 - euporie.menu.contents, 142
 - euporie.menu.item, 142
 - euporie.notebook, 144
 - euporie.output, 146
 - euporie.output.container, 147
 - euporie.output.control, 150
 - euporie.palette, 152
 - euporie.scroll, 154
 - euporie.style, 158
 - euporie.suggest, 158
 - euporie.tab, 159
 - euporie.terminal, 160
 - euporie.text, 166
 - mouse_handler() (euporie.margins.ScrollbarMargin method), 124
 - mouse_handler() (euporie.output.control.AnsiControl method), 152
 - mouse_handler() (euporie.palette.CommandMenuControl method), 153
 - mouse_handler() (euporie.scroll.ScrollbarControl method), 156
 - mouse_scroll_handler() (euporie.scroll.ScrollingContainer method), 156
 - move() (euporie.notebook.Notebook method), 146
 - move_cells_down() (in module euporie.commands.notebook), 83, 86
 - move_cells_up() (in module euporie.commands.notebook), 83, 86
 - move_cursor_down() (euporie.output.control.AnsiControl method), 152
 - move_cursor_down() (euporie.palette.CommandMenuControl method), 153
 - move_cursor_down() (euporie.scroll.ScrollbarControl method), 156
 - move_cursor_left() (in module euporie.commands.buffer), 68, 71
 - move_cursor_right() (in module euporie.commands.buffer), 68, 72
 - move_cursor_up() (euporie.output.control.AnsiControl method), 152
 - move_cursor_up() (euporie.palette.CommandMenuControl method), 153
 - move_cursor_up() (euporie.scroll.ScrollbarControl method), 156
 - move_line() (in module euporie.commands.buffer), 68, 72
 - move_lines_down() (in module euporie.commands.buffer), 68, 72
 - move_lines_up() (in module euporie.commands.buffer), 69, 72
- ## N
- name (euporie.log.QueueHandler property), 122
 - new_line (euporie.markdown.LatexBlock attribute), 137
 - new_line (euporie.markdown.LatexElement attribute), 137
 - new_line (euporie.markdown.LatexInline attribute), 138
 - new_line (euporie.markdown.rich.LatexBlock attribute), 132
 - new_line (euporie.markdown.rich.LatexElement attribute), 132
 - new_line (euporie.markdown.rich.LatexInline attribute), 133
 - new_line (euporie.markdown.rich.Table attribute), 134
 - new_line (euporie.markdown.Table attribute), 140
 - new_notebook() (in module euporie.commands.tui), 90, 91
 - newline() (in module euporie.commands.buffer), 69, 72
 - next_child() (in module euporie.commands.notebook), 83, 86
 - next_tab() (in module euporie.commands.tui), 90, 91
 - NO_COLOR, 34
 - node_to_md() (euporie.markdown.rich.Table static method), 134

- node_to_md() (*euporie.markdown.Table* static method), 140
- NONE (*euporie.box.RoundBorder* attribute), 57
- NONE (*euporie.box.SquareBorder* attribute), 57
- Notebook (class in *euporie.notebook*), 145
- notebook (*euporie.app.base.EuporieApp* property), 40
- notebook (*euporie.app.dump.DumpApp* property), 45
- notebook (*euporie.app.tui.TuiApp* property), 51
- notebook_class (*euporie.app.base.EuporieApp* attribute), 40
- notebook_class (*euporie.app.dump.DumpApp* attribute), 45
- notebook_class (*euporie.app.tui.TuiApp* attribute), 51
- NotebookKernel (class in *euporie.kernel*), 112
- ## O
- on_child_close() (*euporie.markdown.LatexBlock* method), 137
- on_child_close() (*euporie.markdown.LatexElement* method), 138
- on_child_close() (*euporie.markdown.LatexInline* method), 138
- on_child_close() (*euporie.markdown.rich.LatexBlock* method), 132
- on_child_close() (*euporie.markdown.rich.LatexElement* method), 132
- on_child_close() (*euporie.markdown.rich.LatexInline* method), 133
- on_child_close() (*euporie.markdown.rich.Table* method), 134
- on_child_close() (*euporie.markdown.Table* method), 140
- on_cursor_position_changed() (*euporie.cell.CellInputTextArea* method), 60
- on_enter() (*euporie.markdown.LatexBlock* method), 137
- on_enter() (*euporie.markdown.LatexElement* method), 138
- on_enter() (*euporie.markdown.LatexInline* method), 139
- on_enter() (*euporie.markdown.rich.LatexBlock* method), 132
- on_enter() (*euporie.markdown.rich.LatexElement* method), 133
- on_enter() (*euporie.markdown.rich.LatexInline* method), 133
- on_enter() (*euporie.markdown.rich.Table* method), 135
- on_enter() (*euporie.markdown.Table* method), 140
- on_leave() (*euporie.markdown.LatexBlock* method), 137
- on_leave() (*euporie.markdown.LatexElement* method), 138
- on_leave() (*euporie.markdown.LatexInline* method), 139
- on_leave() (*euporie.markdown.rich.LatexBlock* method), 132
- on_leave() (*euporie.markdown.rich.LatexElement* method), 133
- on_leave() (*euporie.markdown.rich.LatexInline* method), 134
- on_leave() (*euporie.markdown.rich.Table* method), 135
- on_leave() (*euporie.markdown.Table* method), 140
- on_output() (*euporie.cell.Cell* method), 59
- on_output() (*euporie.cell.InteractiveCell* method), 62
- on_text() (*euporie.markdown.LatexBlock* method), 137
- on_text() (*euporie.markdown.LatexElement* method), 138
- on_text() (*euporie.markdown.LatexInline* method), 139
- on_text() (*euporie.markdown.rich.LatexBlock* method), 132
- on_text() (*euporie.markdown.rich.LatexElement* method), 133
- on_text() (*euporie.markdown.rich.LatexInline* method), 134
- on_text() (*euporie.markdown.rich.Table* method), 135
- on_text() (*euporie.markdown.Table* method), 140
- on_text_changed() (*euporie.cell.CellInputTextArea* method), 60
- open_file() (*euporie.app.base.EuporieApp* method), 40
- open_file() (*euporie.app.dump.DumpApp* method), 45
- open_file() (*euporie.app.tui.TuiApp* method), 51
- open_file() (in module *euporie.commands.tui*), 90, 91
- open_files() (*euporie.app.base.EuporieApp* method), 40
- open_files() (*euporie.app.dump.DumpApp* method), 46
- open_files() (*euporie.app.tui.TuiApp* method), 51
- output (*euporie.terminal.TerminalInfo* attribute), 164
- OutputControl (class in *euporie.output.control*), 151
- outputs (*euporie.cell.Cell* property), 59
- outputs (*euporie.cell.InteractiveCell* property), 62
- OutputWindow (class in *euporie.output.container*), 148, 149
- ## P
- page
command line option, 21
- page_down_cell_output() (in module *euporie.commands.cell_output*), 74, 75
- page_up_cell_output() (in module *euporie.commands.cell_output*), 74, 75
- PagerState (class in *euporie.cell*), 59
- parse() (*euporie.markdown.InlineParser* method), 135
- parse() (*euporie.markdown.inlines.InlineParser* method), 128

`parse()` (*euporie.markdown.Parser* method), 140
`parse()` (*euporie.markdown.parser.Parser* method), 130
`parseAutolink()` (*euporie.markdown.InlineParser* method), 135
`parseAutolink()` (*euporie.markdown.inlines.InlineParser* method), 128
`parseBackslash()` (*euporie.markdown.InlineParser* method), 135
`parseBackslash()` (*euporie.markdown.inlines.InlineParser* method), 128
`parseBackticks()` (*euporie.markdown.InlineParser* method), 135
`parseBackticks()` (*euporie.markdown.inlines.InlineParser* method), 128
`parseBang()` (*euporie.markdown.InlineParser* method), 135
`parseBang()` (*euporie.markdown.inlines.InlineParser* method), 128
`parseCloseBracket()` (*euporie.markdown.InlineParser* method), 135
`parseCloseBracket()` (*euporie.markdown.inlines.InlineParser* method), 128
`parseEntity()` (*euporie.markdown.InlineParser* method), 135
`parseEntity()` (*euporie.markdown.inlines.InlineParser* method), 128
`parseHtmlTag()` (*euporie.markdown.InlineParser* method), 136
`parseHtmlTag()` (*euporie.markdown.inlines.InlineParser* method), 128
`parseInline()` (*euporie.markdown.InlineParser* method), 136
`parseInline()` (*euporie.markdown.inlines.InlineParser* method), 128
`parseInlines()` (*euporie.markdown.InlineParser* method), 136
`parseInlines()` (*euporie.markdown.inlines.InlineParser* method), 128
`parseLinkDestination()` (*euporie.markdown.InlineParser* method), 136
`parseLinkDestination()` (*euporie.markdown.inlines.InlineParser* method), 129
`parseLinkLabel()` (*euporie.markdown.InlineParser* method), 136
`parseLinkLabel()` (*euporie.markdown.inlines.InlineParser* method), 129
`parseLinkTitle()` (*euporie.markdown.InlineParser* method), 136
`parseLinkTitle()` (*euporie.markdown.inlines.InlineParser* method), 129
`parseMath()` (*euporie.markdown.InlineParser* method), 136
`parseMath()` (*euporie.markdown.inlines.InlineParser* method), 129
`parseNewline()` (*euporie.markdown.InlineParser* method), 136
`parseNewline()` (*euporie.markdown.inlines.InlineParser* method), 129
`parseOpenBracket()` (*euporie.markdown.InlineParser* method), 136
`parseOpenBracket()` (*euporie.markdown.inlines.InlineParser* method), 129
`Parser` (class in *euporie.markdown*), 139
`Parser` (class in *euporie.markdown.parser*), 130
`parseReference()` (*euporie.markdown.InlineParser* method), 136
`parseReference()` (*euporie.markdown.inlines.InlineParser* method), 129
`parseString()` (*euporie.markdown.InlineParser* method), 136
`parseString()` (*euporie.markdown.inlines.InlineParser* method), 129
`paste()` (*euporie.notebook.Notebook* method), 146
`paste_cells()` (in module *euporie.commands.notebook*), 83, 86
`paste_clipboard()` (in module *euporie.commands.buffer*), 69, 72
`Pattern` (class in *euporie.box*), 54, 56
`pattern` (*euporie.terminal.BackgroundColor* attribute), 162
`pattern` (*euporie.terminal.ColorQueryMixin* attribute), 162
`pattern` (*euporie.terminal.DepthOfColor* attribute), 162
`pattern` (*euporie.terminal.ForegroundColor* attribute), 163
`pattern` (*euporie.terminal.ItemGraphicsStatus* attribute), 163
`pattern` (*euporie.terminal.KittyGraphicsStatus* attribute), 163
`pattern` (*euporie.terminal.PixelDimensions* attribute), 164
`pattern` (*euporie.terminal.SixelGraphicsStatus* attribute), 164
`pattern` (*euporie.terminal.TerminalQuery* attribute), 165
`peek()` (*euporie.markdown.InlineParser* method), 136
`peek()` (*euporie.markdown.inlines.InlineParser* method), 129
`pil_to_ansi_py_img2unicode()` (in module *eu-*

- `porie.convert.formats.ansi`), 101, 103
`pil_to_ansi_py_timg()` (in module `euporie.convert.formats.ansi`), 101, 103
`pil_to_png_py_pil()` (in module `euporie.convert.formats.png`), 107
`pil_to_sixel_py_timg()` (in module `euporie.convert.formats.sixel`), 108, 109
`pil_to_sixel_py_timg()` (in module `euporie.convert.formats.sixel`), 109
`PixelDimensions` (class in `euporie.terminal`), 161, 164
`png_to_ansi_img2txt()` (in module `euporie.convert.formats.ansi`), 102, 103
`png_to_ansi_py_placeholder()` (in module `euporie.convert.formats.ansi`), 102, 103
`png_to_pil_py()` (in module `euporie.convert.formats.pil`), 106
`png_to_sixel_img2sixel()` (in module `euporie.convert.formats.sixel`), 109
`poll()` (`euporie.kernel.NotebookKernel` method), 114
`pos` (`euporie.markdown.InlineParser` attribute), 136
`pos` (`euporie.markdown.inlines.InlineParser` attribute), 129
`post_load()` (`euporie.app.base.EuporieApp` method), 40
`post_load()` (`euporie.app.dump.DumpApp` method), 46
`post_load()` (`euporie.app.tui.TuiApp` method), 51
`pre_exit()` (`euporie.app.dump.DumpApp` method), 46
`pre_run()` (`euporie.app.base.EuporieApp` method), 40
`pre_run()` (`euporie.app.dump.DumpApp` method), 46
`pre_run()` (`euporie.app.tui.TuiApp` method), 51
`pre_run_callable` (`euporie.app.dump.DumpApp` attribute), 46
`pre_run_callable` (`euporie.app.tui.TuiApp` attribute), 51
`preferred_height()` (`euporie.box.BorderLine` method), 55
`preferred_height()` (`euporie.box.Pattern` method), 56
`preferred_height()` (`euporie.containers.PrintingContainer` method), 95
`preferred_height()` (`euporie.output.container.GraphicWindow` method), 149
`preferred_height()` (`euporie.output.container.OutputWindow` method), 149
`preferred_height()` (`euporie.output.control.AnsiControl` method), 152
`preferred_height()` (`euporie.palette.CommandMenuControl` method), 153
`preferred_height()` (`euporie.scroll.ScrollbarControl` method), 156
`preferred_height()` (`euporie.scroll.ScrollingContainer` method), 156
`preferred_width()` (`euporie.box.BorderLine` method), 55
`preferred_width()` (`euporie.box.Pattern` method), 56
`preferred_width()` (`euporie.containers.PrintingContainer` method), 95
`preferred_width()` (`euporie.output.container.GraphicWindow` method), 149
`preferred_width()` (`euporie.output.container.OutputWindow` method), 149
`preferred_width()` (`euporie.output.control.AnsiControl` method), 152
`preferred_width()` (`euporie.palette.CommandMenuControl` method), 153
`preferred_width()` (`euporie.scroll.ScrollbarControl` method), 156
`preferred_width()` (`euporie.scroll.ScrollingContainer` method), 157
`prefix` (`euporie.menu.item.MenuItem` property), 143
`prefix` (`euporie.menu.MenuItem` property), 144
`prefix_width` (`euporie.menu.item.MenuItem` property), 143
`prefix_width` (`euporie.menu.MenuItem` property), 144
`previous_tab()` (in module `euporie.commands.tui`), 90, 91
`print_text()` (`euporie.app.base.EuporieApp` method), 40
`print_text()` (`euporie.app.dump.DumpApp` method), 46
`print_text()` (`euporie.app.tui.TuiApp` method), 51
`PrintingContainer` (class in `euporie.containers`), 95
`process_default_iopub_rsp()` (`euporie.kernel.NotebookKernel` method), 114
`process_inlines()` (`euporie.markdown.Parser` method), 140
`process_inlines()` (`euporie.markdown.parser.Parser` method), 130
`processEmphasis()` (`euporie.markdown.InlineParser` method), 136
`processEmphasis()` (`euporie.markdown.inlines.InlineParser` method), 129
`prompt` (`euporie.cell.Cell` property), 60
`prompt` (`euporie.cell.InteractiveCell` property), 62
- ## Q
- `QueueHandler` (class in `euporie.log`), 121

`quit()` (in module *euporie.commands.tui*), 90, 91
`quoted_insert` (*euporie.app.base.EuporieApp* attribute), 40
`quoted_insert` (*euporie.app.dump.DumpApp* attribute), 46
`quoted_insert` (*euporie.app.tui.TuiApp* attribute), 52

R

`ran()` (*euporie.cell.Cell* method), 60
`ran()` (*euporie.cell.InteractiveCell* method), 62
`redo()` (in module *euporie.commands.buffer*), 69, 72
`reformat()` (*euporie.cell.Cell* method), 60
`reformat()` (*euporie.cell.InteractiveCell* method), 62
`reformat_cells()` (in module *euporie.commands.notebook*), 83, 86
`reformat_notebook()` (in module *euporie.commands.notebook*), 83, 86
`refresh()` (*euporie.app.base.EuporieApp* method), 40
`refresh()` (*euporie.app.dump.DumpApp* method), 46
`refresh()` (*euporie.app.tui.TuiApp* method), 52
`refresh()` (*euporie.notebook.Notebook* method), 146
`register()` (*euporie.terminal.TerminalInfo* method), 165
`register()` (in module *euporie.convert.base*), 97
`release()` (*euporie.log.QueueHandler* method), 123
`remove()` (*euporie.keys.KeyBindingsInfo* method), 119
`remove_binding()` (*euporie.keys.KeyBindingsInfo* method), 120
`remove_float()` (*euporie.app.base.EuporieApp* method), 40
`remove_float()` (*euporie.app.dump.DumpApp* method), 46
`remove_float()` (*euporie.app.tui.TuiApp* method), 52
`remove_output_graphic_floats()` (*euporie.cell.Cell* method), 60
`remove_output_graphic_floats()` (*euporie.cell.InteractiveCell* method), 62
`removeBracket()` (*euporie.markdown.InlineParser* method), 136
`removeBracket()` (*euporie.markdown.inlines.InlineParser* method), 129
`removeDelimiter()` (*euporie.markdown.InlineParser* method), 136
`removeDelimiter()` (*euporie.markdown.inlines.InlineParser* method), 129
`removeDelimitersBetween()` (*euporie.markdown.InlineParser* static method), 136
`removeDelimitersBetween()` (*euporie.markdown.inlines.InlineParser* static method), 129

`removeFilter()` (*euporie.log.QueueHandler* method), 123
`render()` (*euporie.log.LogView* method), 121
`render()` (*euporie.scroll.ChildRenderInfo* method), 155
`render_counter` (*euporie.app.base.EuporieApp* attribute), 40
`render_counter` (*euporie.app.dump.DumpApp* attribute), 46
`render_counter` (*euporie.app.tui.TuiApp* attribute), 52
`render_outputs()` (*euporie.cell.Cell* method), 60
`render_outputs()` (*euporie.cell.InteractiveCell* method), 62
`rendered_cells()` (*euporie.notebook.Notebook* method), 146
`repeat()` (*euporie.margins.ScrollbarMargin* method), 124
`REPLACE` (*euporie.key_binding.micro_state.InputMode* attribute), 117
`replace_selection()` (in module *euporie.commands.buffer*), 69, 72
`reset()` (*euporie.app.base.EuporieApp* method), 40
`reset()` (*euporie.app.dump.DumpApp* method), 46
`reset()` (*euporie.app.tui.TuiApp* method), 52
`reset()` (*euporie.box.BorderLine* method), 55
`reset()` (*euporie.box.Pattern* method), 56
`reset()` (*euporie.containers.PrintingContainer* method), 96
`reset()` (*euporie.key_binding.micro_state.MicroState* method), 117
`reset()` (*euporie.output.container.GraphicWindow* method), 149
`reset()` (*euporie.output.container.OutputWindow* method), 149
`reset()` (*euporie.output.control.AnsiControl* method), 152
`reset()` (*euporie.palette.CommandMenuControl* method), 153
`reset()` (*euporie.scroll.ScrollbarControl* method), 156
`reset()` (*euporie.scroll.ScrollingContainer* method), 157
`reset()` (*euporie.terminal.Vt100Parser* method), 165
`restart()` (*euporie.kernel.NotebookKernel* method), 114
`restart_()` (*euporie.kernel.NotebookKernel* method), 114
`restart_kernel()` (in module *euporie.commands.notebook*), 83, 86
`RoundBorder` (class in *euporie.box*), 55, 56
`run`
 command line option, 21
`run()` (*euporie.app.base.EuporieApp* method), 41
`run()` (*euporie.app.dump.DumpApp* method), 46
`run()` (*euporie.app.tui.TuiApp* method), 52
`run()` (*euporie.commands.base.Command* method), 65

- `run()` (*euporie.commands.Command* method), 92
`run()` (*euporie.kernel.NotebookKernel* method), 114
`run_()` (*euporie.kernel.NotebookKernel* method), 114
`run_after_external_edit`
 command line option, 22
`run_after_external_edit()` (in module *euporie.commands.config*), 77, 79
`run_all_cells()` (in module *euporie.commands.notebook*), 83, 86
`run_async()` (*euporie.app.base.EuporieApp* method), 41
`run_async()` (*euporie.app.dump.DumpApp* method), 47
`run_async()` (*euporie.app.tui.TuiApp* method), 52
`run_cell()` (*euporie.notebook.Notebook* method), 146
`run_cell_and_insert_below()` (in module *euporie.commands.notebook*), 84, 86
`run_macro()` (in module *euporie.commands.buffer*), 69, 72
`run_or_render()` (*euporie.cell.Cell* method), 60
`run_or_render()` (*euporie.cell.InteractiveCell* method), 62
`run_selected_cells()` (in module *euporie.commands.notebook*), 84, 86
`run_selected_cells_and_select_next_cell()` (in module *euporie.commands.notebook*), 84, 86
`run_system_command()` (*euporie.app.base.EuporieApp* method), 41
`run_system_command()` (*euporie.app.dump.DumpApp* method), 47
`run_system_command()` (*euporie.app.tui.TuiApp* method), 53
- ## S
- `save()` (*euporie.notebook.Notebook* method), 146
`save_notebook()` (in module *euporie.commands.notebook*), 84, 86
`scanDelims()` (*euporie.markdown.InlineParser* method), 136
`scanDelims()` (*euporie.markdown.inlines.InlineParser* method), 129
`scroll()` (*euporie.scroll.ScrollingContainer* method), 157
`scroll_down()` (in module *euporie.commands.notebook*), 84, 86
`scroll_down_5_lines()` (in module *euporie.commands.notebook*), 84, 86
`scroll_down_cell_output()` (in module *euporie.commands.cell_output*), 74, 75
`scroll_to()` (*euporie.scroll.ScrollingContainer* method), 157
`scroll_up()` (in module *euporie.commands.notebook*), 84, 86
`scroll_up_5_lines()` (in module *euporie.commands.notebook*), 84, 86
`scroll_up_cell_output()` (in module *euporie.commands.cell_output*), 75
`ScrollbarControl` (class in *euporie.scroll*), 155
`ScrollbarMargin` (class in *euporie.margins*), 123
`ScrollingContainer` (class in *euporie.scroll*), 155, 156
`select()` (*euporie.cell.InteractiveCell* method), 62
`select()` (*euporie.palette.CommandPalette* method), 154
`select_5th_next_cell()` (in module *euporie.commands.notebook*), 85, 86
`select_5th_previous_cell()` (in module *euporie.commands.notebook*), 85, 87
`select_all()` (in module *euporie.commands.buffer*), 69, 72
`select_all_cells()` (in module *euporie.commands.notebook*), 85, 87
`select_first_cell()` (in module *euporie.commands.notebook*), 85, 87
`select_last_cell()` (in module *euporie.commands.notebook*), 85, 87
`select_previous_cell()` (in module *euporie.commands.notebook*), 85, 87
`selected` (*euporie.cell.Cell* property), 60
`selected` (*euporie.cell.InteractiveCell* property), 62
`selected_indices` (*euporie.scroll.ScrollingContainer* property), 157
`selected_slice` (*euporie.scroll.ScrollingContainer* property), 157
`send()` (*euporie.terminal.BackgroundColor* method), 162
`send()` (*euporie.terminal.DepthOfColor* method), 162
`send()` (*euporie.terminal.ForegroundColor* method), 163
`send()` (*euporie.terminal.ItemGraphicsStatus* method), 163
`send()` (*euporie.terminal.KittyGraphicsStatus* method), 163
`send()` (*euporie.terminal.PixelDimensions* method), 164
`send()` (*euporie.terminal.SixelGraphicsStatus* method), 164
`send()` (*euporie.terminal.TerminalQuery* method), 165
`send_all()` (*euporie.terminal.TerminalInfo* method), 165
`set_background()` (in module *euporie.convert.formats.pil*), 106
`set_cell_type()` (*euporie.cell.Cell* method), 60
`set_cell_type()` (*euporie.cell.InteractiveCell* method), 62
`set_edit_mode()` (*euporie.app.base.EuporieApp* method), 42
`set_edit_mode()` (*euporie.app.dump.DumpApp* method), 47
`set_edit_mode()` (*euporie.app.tui.TuiApp* method), 53
`set_edit_mode()` (in module *euporie.commands.notebook*), 84, 86

- porie.commands.config*), 77, 79
- `set_metadata()` (*euporie.kernel.NotebookKernel* method), 114
- `set_name()` (*euporie.log.QueueHandler* method), 123
- `setext_heading()` (*euporie.markdown.blocks.BlockStarts* static method), 127
- `setFormatter()` (*euporie.log.QueueHandler* method), 123
- `setLevel()` (*euporie.log.QueueHandler* method), 123
- `setup_logs()` (in module *euporie.log*), 120, 123
- `shortcut_str` (*euporie.menu.item.MenuItem* property), 143
- `shortcut_str` (*euporie.menu.MenuItem* property), 144
- `show()` (*euporie.palette.CommandPalette* method), 154
- `show_cell_borders`
command line option, 25
- `show_cell_borders()` (in module *euporie.commands.config*), 77, 79
- `show_command_palette()` (in module *euporie.commands.tui*), 90, 91
- `show_contextual_help()` (in module *euporie.commands.buffer*), 69, 72
- `show_line_numbers()` (in module *euporie.commands.config*), 78, 79
- `show_status_bar`
command line option, 24
- `show_status_bar()` (in module *euporie.commands.config*), 78, 79
- `shutdown()` (*euporie.kernel.NotebookKernel* method), 114
- `shutdown_()` (*euporie.kernel.NotebookKernel* method), 114
- `SixelGraphicControl` (class in *euporie.output.control*), 151
- `SixelGraphicsStatus` (class in *euporie.terminal*), 161, 164
- `size()` (*euporie.output.control.AnsiControl* method), 152
- `specs` (*euporie.kernel.NotebookKernel* property), 114
- `split()` (*euporie.cell.InteractiveCell* method), 62
- `SPLIT_BOTTOM` (*euporie.box.RoundBorder* attribute), 57
- `SPLIT_BOTTOM` (*euporie.box.SquareBorder* attribute), 57
- `split_cell()` (*euporie.notebook.Notebook* method), 146
- `split_cell()` (in module *euporie.commands.cell*), 73
- `SPLIT_LEFT` (*euporie.box.RoundBorder* attribute), 57
- `SPLIT_LEFT` (*euporie.box.SquareBorder* attribute), 57
- `SPLIT_RIGHT` (*euporie.box.RoundBorder* attribute), 57
- `SPLIT_RIGHT` (*euporie.box.SquareBorder* attribute), 57
- `SPLIT_TOP` (*euporie.box.RoundBorder* attribute), 57
- `SPLIT_TOP` (*euporie.box.SquareBorder* attribute), 57
- `spnl()` (*euporie.markdown.InlineParser* method), 136
- `spnl()` (*euporie.markdown.inlines.InlineParser* method), 129
- `SquareBorder` (class in *euporie.box*), 55, 57
- `start()` (*euporie.kernel.NotebookKernel* method), 115
- `start_()` (*euporie.kernel.NotebookKernel* method), 115
- `start_macro()` (*euporie.key_binding.micro_state.MicroState* method), 117
- `start_macro()` (in module *euporie.commands.buffer*), 70, 72
- `start_selection()` (in module *euporie.commands.buffer*), 70, 72
- `status` (*euporie.kernel.NotebookKernel* property), 115
- `statusbar_fields()` (*euporie.log.LogView* method), 121
- `statusbar_fields()` (*euporie.menu.bar.MenuContainer* method), 141
- `statusbar_fields()` (*euporie.menu.MenuContainer* method), 143
- `statusbar_fields()` (*euporie.notebook.Notebook* method), 146
- `statusbar_fields()` (*euporie.palette.CommandPalette* method), 154
- `statusbar_fields()` (*euporie.tab.Tab* method), 159
- `stdout_to_log` (class in *euporie.log*), 121
- `stop()` (*euporie.kernel.NotebookKernel* method), 115
- `stop_()` (*euporie.kernel.NotebookKernel* method), 115
- `style_name` (*euporie.markdown.LatexBlock* attribute), 137
- `style_name` (*euporie.markdown.LatexElement* attribute), 138
- `style_name` (*euporie.markdown.LatexInline* attribute), 139
- `style_name` (*euporie.markdown.rich.LatexBlock* attribute), 132
- `style_name` (*euporie.markdown.rich.LatexElement* attribute), 133
- `style_name` (*euporie.markdown.rich.LatexInline* attribute), 134
- `style_name` (*euporie.markdown.rich.Table* attribute), 135
- `style_name` (*euporie.markdown.Table* attribute), 141
- `suffix` (*euporie.menu.item.MenuItem* property), 143
- `suffix` (*euporie.menu.MenuItem* property), 144
- `suffix_width` (*euporie.menu.item.MenuItem* property), 143
- `suffix_width` (*euporie.menu.MenuItem* property), 144
- `suspend_to_background()` (*euporie.app.base.EuporieApp* method), 42
- `suspend_to_background()` (*euporie.app.dump.DumpApp* method), 47
- `suspend_to_background()` (*euporie.app.tui.TuiApp* method), 53
- `svg_to_png_py_cairosvg()` (in module *eu-*

porie.convert.formats.png), 107
 switch_background_pattern() (in module *euporie.commands.config*), 78, 79
 syntax_theme
 command line option, 26

T

Tab (class in *euporie.tab*), 159
 tab (*euporie.app.base.EuporieApp* property), 42
 tab (*euporie.app.dump.DumpApp* property), 47
 tab (*euporie.app.tui.TuiApp* property), 53
 tab_container() (*euporie.app.tui.TuiApp* method), 53
 tab_idx (*euporie.app.base.EuporieApp* property), 42
 tab_idx (*euporie.app.dump.DumpApp* property), 48
 tab_idx (*euporie.app.tui.TuiApp* property), 53
 tab_size
 command line option, 22
 Table (class in *euporie.markdown*), 140
 Table (class in *euporie.markdown.blocks*), 127
 Table (class in *euporie.markdown.blocks.tables*), 126
 Table (class in *euporie.markdown.rich*), 131, 134
 table() (*euporie.markdown.blocks.BlockStarts* static method), 127
 tabs (*euporie.app.dump.DumpApp* attribute), 48
 tabs (*euporie.app.tui.TuiApp* attribute), 53
 terminal_polling_interval
 command line option, 22
 terminal_size_px (*euporie.terminal.TerminalInfo* property), 165
 TerminalInfo (class in *euporie.terminal*), 161, 164
 TerminalQuery (class in *euporie.terminal*), 161, 165
 text (*euporie.cell.CellInputTextArea* property), 60
 text (*euporie.cell.CellStdinTextArea* property), 61
 text (*euporie.menu.item.MenuItem* property), 143
 text (*euporie.menu.MenuItem* property), 144
 text (*euporie.text.FormattedTextArea* property), 167
 text_changed() (*euporie.palette.CommandPalette* method), 154
 thematic_break() (*euporie.markdown.blocks.BlockStarts* static method), 127
 ThickVerticalBorder (class in *euporie.box*), 55
 timeoutlen (*euporie.app.base.EuporieApp* attribute), 42
 timeoutlen (*euporie.app.dump.DumpApp* attribute), 48
 timeoutlen (*euporie.app.tui.TuiApp* attribute), 53
 title (*euporie.log.LogView* property), 121
 title (*euporie.notebook.Notebook* property), 146
 title (*euporie.tab.Tab* property), 159
 tmux_graphics
 command line option, 21
 tmux_terminal_graphics() (in module *euporie.commands.config*), 78, 79
 tmuxify() (in module *euporie.terminal*), 160, 165

to_formatted_text() (*euporie.keys.KeyBindingsInfo* class method), 120
 toggle() (*euporie.config.Config* method), 94
 toggle() (*euporie.palette.CommandPalette* method), 154
 toggle_case() (in module *euporie.commands.buffer*), 70, 72
 toggle_comment() (in module *euporie.commands.buffer*), 70, 72
 toggle_overwrite_mode() (in module *euporie.commands.buffer*), 70, 72
 TOP_LEFT (*euporie.box.RoundBorder* attribute), 57
 TOP_LEFT (*euporie.box.SquareBorder* attribute), 57
 TOP_RIGHT (*euporie.box.RoundBorder* attribute), 57
 TOP_RIGHT (*euporie.box.SquareBorder* attribute), 57
 trigger_refresh() (*euporie.cell.Cell* method), 60
 trigger_refresh() (*euporie.cell.InteractiveCell* method), 62
 ttimeoutlen (*euporie.app.base.EuporieApp* attribute), 42
 ttimeoutlen (*euporie.app.dump.DumpApp* attribute), 48
 ttimeoutlen (*euporie.app.tui.TuiApp* attribute), 53
 TuiApp (class in *euporie.app.tui*), 48
 TuiNotebook (class in *euporie.notebook*), 145
 type_key() (in module *euporie.commands.buffer*), 70, 72

U

undo() (in module *euporie.commands.buffer*), 70, 72
 unhook() (*euporie.log.QueueHandler* class method), 123
 unindent_lines() (in module *euporie.commands.buffer*), 70, 72
 unshift_move() (in module *euporie.commands.buffer*), 71, 72
 update_color_scheme() (in module *euporie.commands.config*), 78, 79
 update_style() (*euporie.app.base.EuporieApp* method), 42
 update_style() (*euporie.app.dump.DumpApp* method), 48
 update_style() (*euporie.app.tui.TuiApp* method), 54
 update_syntax_theme() (in module *euporie.commands.config*), 78, 79
 use_full_width() (in module *euporie.commands.config*), 78, 79

V

valid_user (*euporie.config.Config* attribute), 94
 validate_slice() (*euporie.scroll.ScrollingContainer* method), 157
 value (*euporie.key_binding.micro_state.InputMode* attribute), 117

`value` (*euporie.terminal.BackgroundColor* property), 162
`value` (*euporie.terminal.DepthOfColor* property), 162
`value` (*euporie.terminal.ForegroundColor* property), 163
`value` (*euporie.terminal.ItermGraphicsStatus* property), 163
`value` (*euporie.terminal.KittyGraphicsStatus* property), 163
`value` (*euporie.terminal.PixelDimensions* property), 164
`value` (*euporie.terminal.SixelGraphicsStatus* property), 164
`value` (*euporie.terminal.TerminalQuery* property), 165
`verify()` (*euporie.terminal.BackgroundColor* method), 162
`verify()` (*euporie.terminal.ColorQueryMixin* method), 162
`verify()` (*euporie.terminal.DepthOfColor* method), 162
`verify()` (*euporie.terminal.ForegroundColor* method), 163
`verify()` (*euporie.terminal.ItermGraphicsStatus* method), 163
`verify()` (*euporie.terminal.KittyGraphicsStatus* method), 164
`verify()` (*euporie.terminal.PixelDimensions* method), 164
`verify()` (*euporie.terminal.SixelGraphicsStatus* method), 164
`verify()` (*euporie.terminal.TerminalQuery* method), 165
`version`
 command line option, 20
`VERTICAL` (*euporie.box.RoundBorder* attribute), 57
`VERTICAL` (*euporie.box.SquareBorder* attribute), 57
`vi_state` (*euporie.app.base.EuporieApp* attribute), 42
`vi_state` (*euporie.app.dump.DumpApp* attribute), 48
`vi_state` (*euporie.app.tui.TuiApp* attribute), 54
`view_documentation()` (in module *euporie.commands.tui*), 90, 91
`view_logs()` (in module *euporie.commands.tui*), 91
`Vt100Parser` (class in *euporie.terminal*), 162, 165

W

`weight` (*euporie.convert.base.Convertor* attribute), 97
`width` (*euporie.menu.item.MenuItem* property), 143
`width` (*euporie.menu.MenuItem* property), 144
`window_render_info` (*euporie.margins.ScrollbarMargin* attribute), 124
`wrap_selection_cmd()` (in module *euporie.commands.buffer*), 71, 73
`write_to_screen()` (*euporie.box.BorderLine* method), 55
`write_to_screen()` (*euporie.box.Pattern* method), 56

`write_to_screen()` (*euporie.containers.PrintingContainer* method), 96
`write_to_screen()` (*euporie.output.container.GraphicWindow* method), 149
`write_to_screen()` (*euporie.output.container.OutputWindow* method), 149
`write_to_screen()` (*euporie.scroll.ScrollingContainer* method), 157