# euporie

**Josiah Outram Halstead**

**May 13, 2024**

# CONTENTS:

**Euporie is a terminal based interactive computing environment for Jupyter.**

Euporie's apps allow you to interact with Jupyter kernels, and run Jupyter notebooks - entirely from the terminal.

If you're working with Jupyter notebooks in a terminal only environment, like an SSH server or a container, or just prefer working in the terminal, then euporie is the tool for you!

| Console | Notebook | Preview | Hub |
|---|---|---|---|


View more screenshots here

CHAPTER

ONE

# 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.

CHAPTER

TWO

# FEATURES

- Edit and run notebooks in the terminal
- Run code interactively in a console
- Display images using terminal graphics (sixel / iterm / kitty)
- Use Jupyter widgets interactively in the terminal
- Render rich kernel output (markdown, tables, images, LaTeX, HTML, SVG, & PDF)
- Tab-completion, line suggestions and contextual help
- Convert a console session to a notebook
- Micro / Vim / Emacs style key-bindings

CHAPTER

THREE

# USAGE

### Notebooks

You can edit a notebook using `euporie-notebook`, and passing the notebook's file path or URI as a command line argument:

```
$ euporie-notebook notebook.ipynb
```

Alternatively, launch `euporie-notebooks` and open a notebook file by selecting "Open" from the file menu (`Ctrl+O`).

### Console

To connect to a Jupyter kernel and run code interactively in a console session, you can run

```
$ euporie-console
```

(You can press `Ctrl+C` to open the command palette in `euporie-console`).

### Preview

To preview a notebook to the terminal, use the `euporie-preview` subcommand:

```
$ euporie-preview notebook.ipynb
```

### Hub

To run euporie hub, a multi-user SSH server for euporie apps, run:

```
$ euporie-hub --port 8022 --host-keys=ssh_host_ed25519_key --client-
↪keys=authorized_keys
```

where `ssh_host_ed25519_key` is the path to your host key file, and `authorized_keys` is a file containing SSH public keys allowed to connect.

CHAPTER

FOUR

# DOCUMENTATION

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

The code is available on GitHub at: https://github.com/joouha/euporie

CHAPTER

FIVE

# COMPATIBILITY

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

## 5.1 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@dev
```

**Note:** Although euporie does not have any compiled components, some of its dependencies may require compilation as part of their build process, depending on the availability of binary wheels. If this is the case, you may need to install the relevant build dependencies for your distribution, such as *python-dev* and *gcc* or equivalent.

### 5.1.1 Try without installing

You can use **`pipx`** to try euporie before installing it:

```
$ pipx run --spec 'euporie[all]' euporie notebook
```

You can also try euporie online here:

https://mybinder.org/v2/gh/joouha/euporie-binder/HEAD?urlpath=%2Feuporie%2F

### 5.1.2 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.

This is supported by kitty, WezTerm, and Konsole.

#### 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
  - catimg
  - 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

## 5.2 Gallery

This page showcases screenshots demonstrating some of the features of euporie

### 5.2.1 v2.x.x

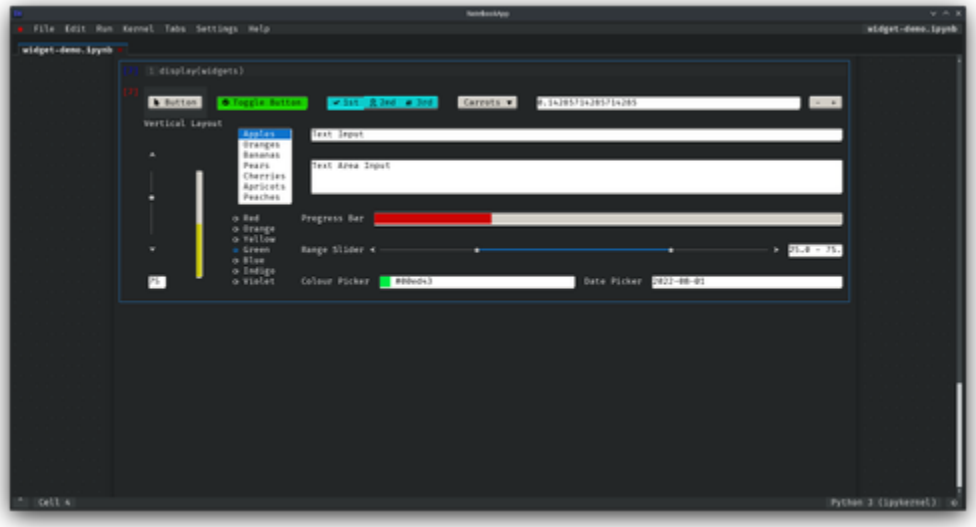


Fig. 1: Ipywidget support

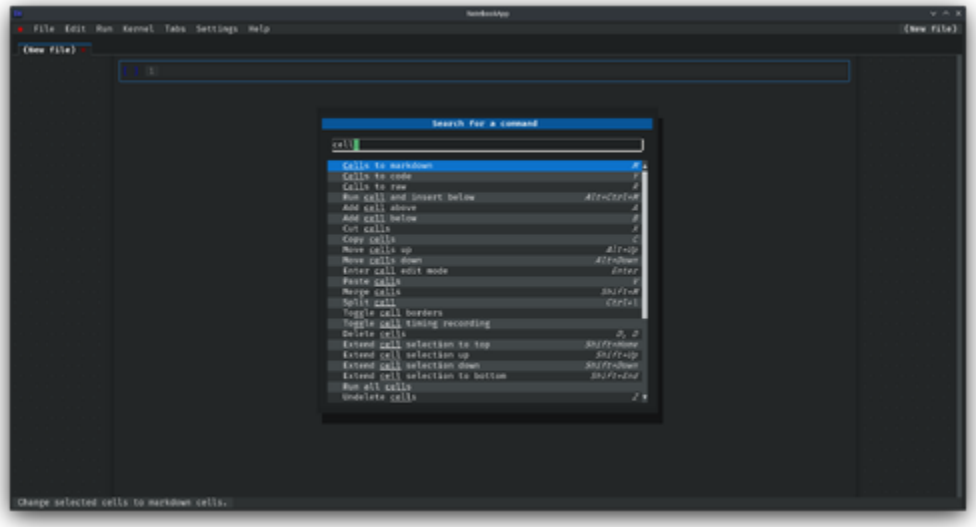


Fig. 2: Command palette

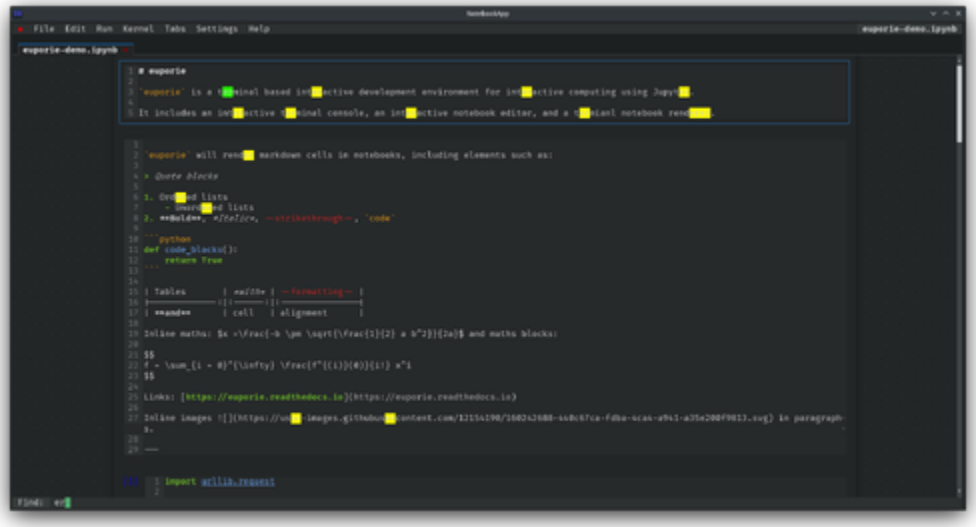


Fig. 3: Search

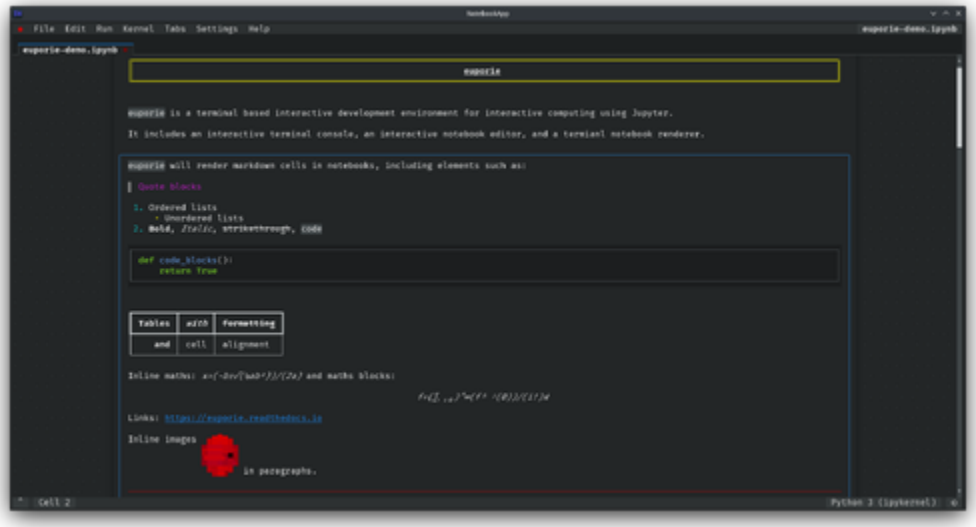


Fig. 4: Markdown cell rendering

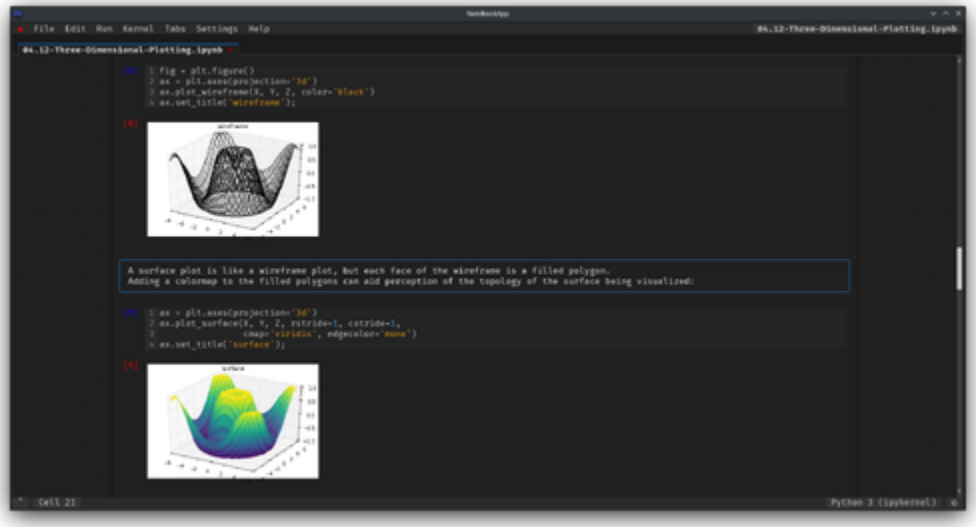


Fig. 5: Dark color scheme

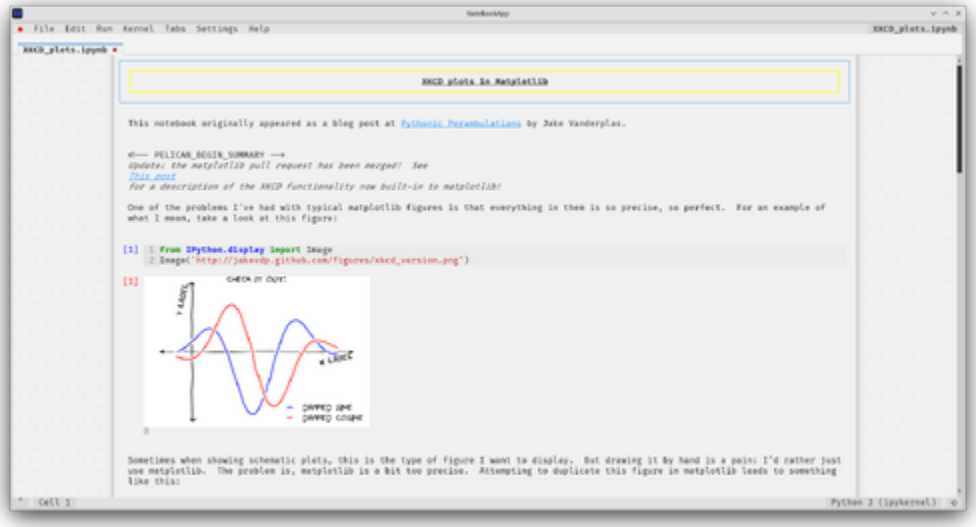


Fig. 6: Light color scheme

Fig. 7: Black color scheme

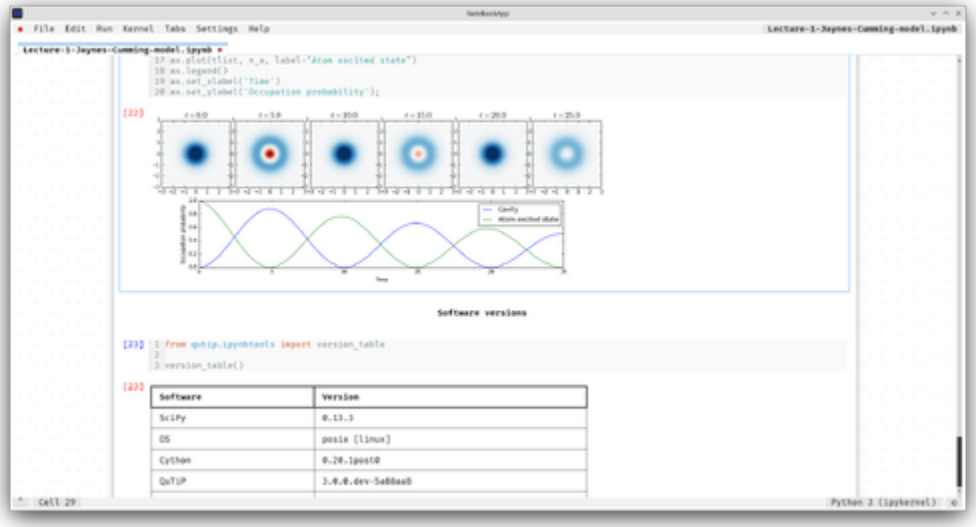


Fig. 8: White color scheme

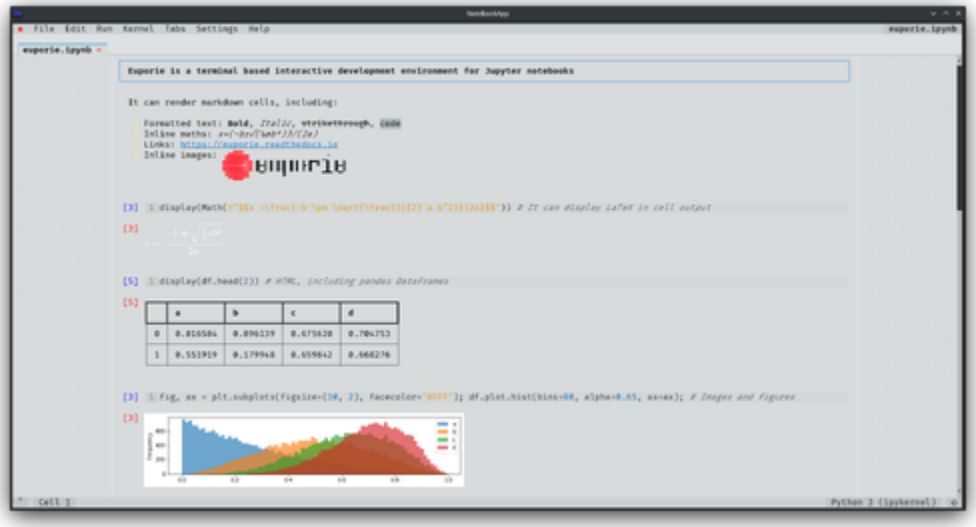


Fig. 9: Inverse color scheme

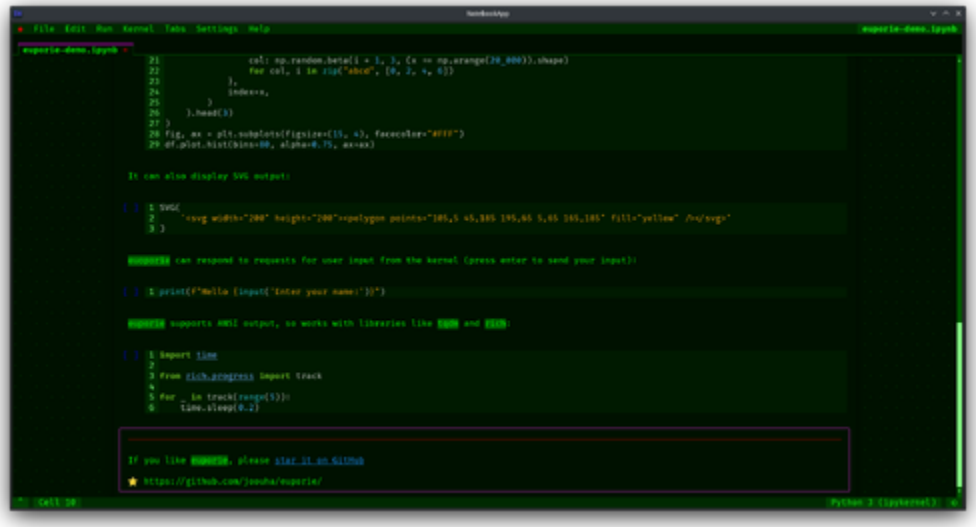


Fig. 10: Custom color schemes

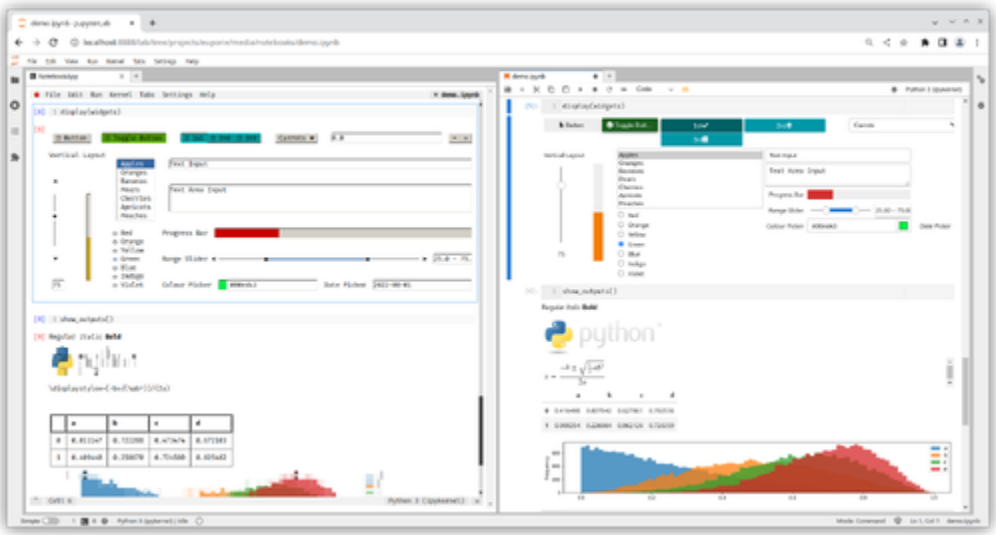


Fig. 11: Running in a terminal in Jupyter Lab

### 5.2.2 v1.x.x

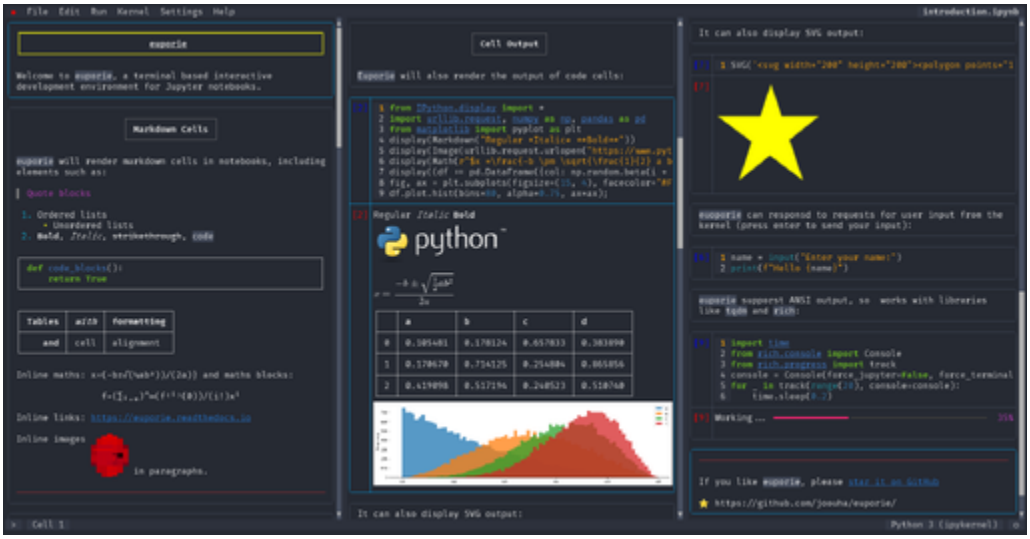


Fig. 12: Displaying multiple notebooks side-by-side

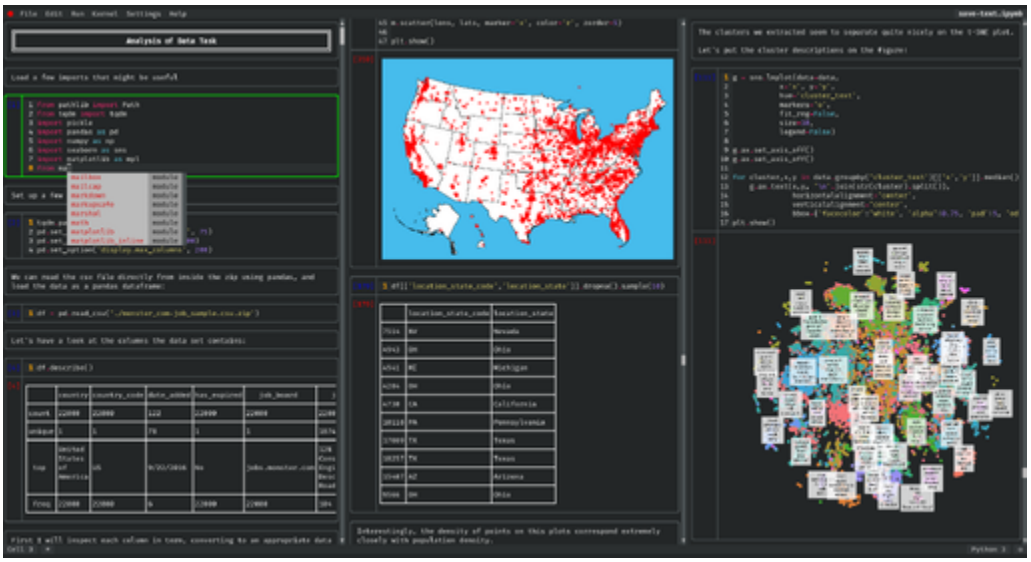


Fig. 13: Tab completion

Fig. 14: Displaying contextual help

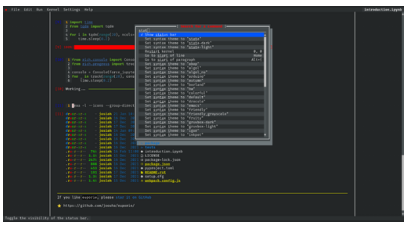


Fig. 15: Command Palette

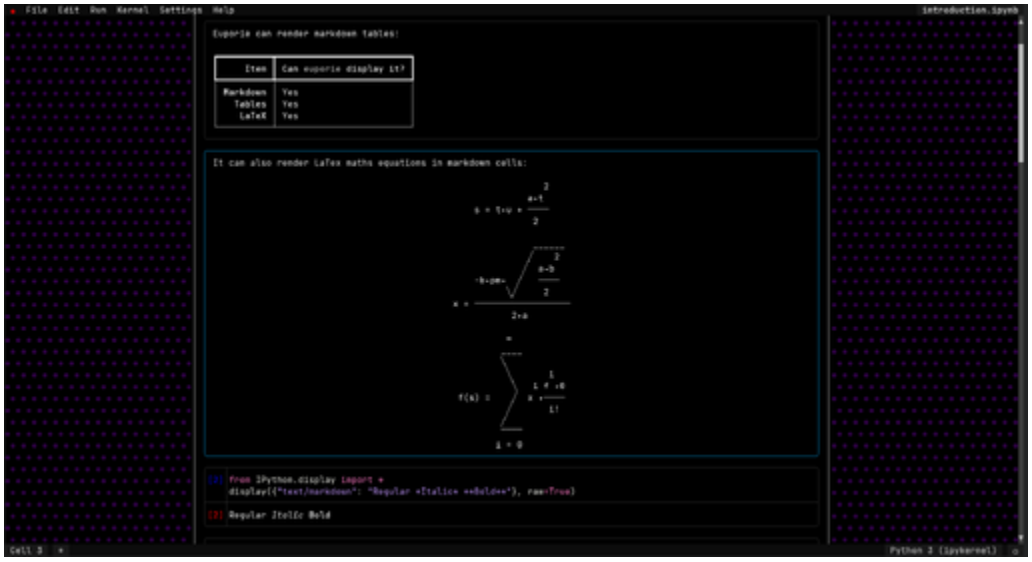


Fig. 16: Display LaTeX formulae in markdown cells with SymPy

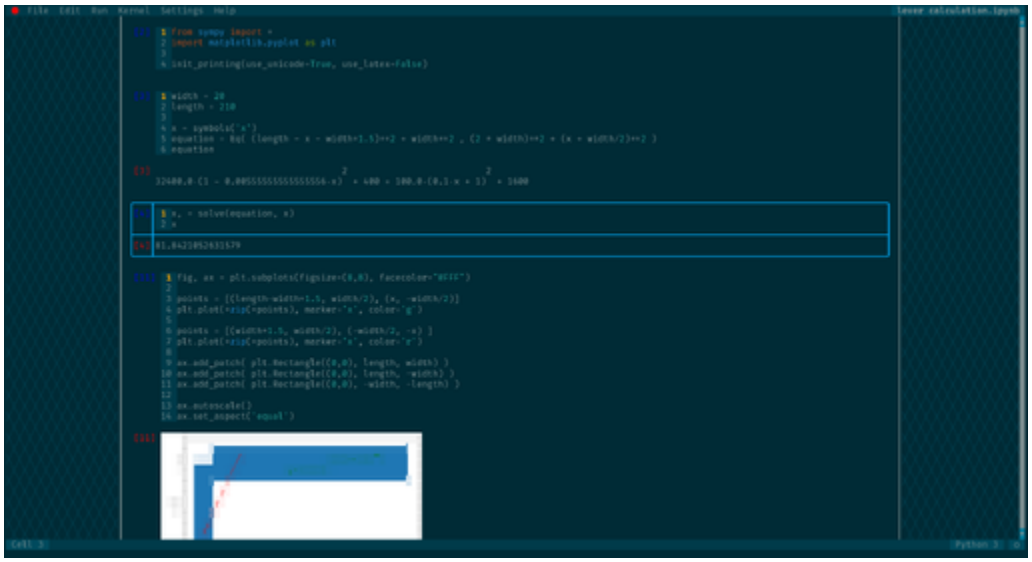


Fig. 17: Adapting to terminal colour theme

Fig. 18: Running on Windows

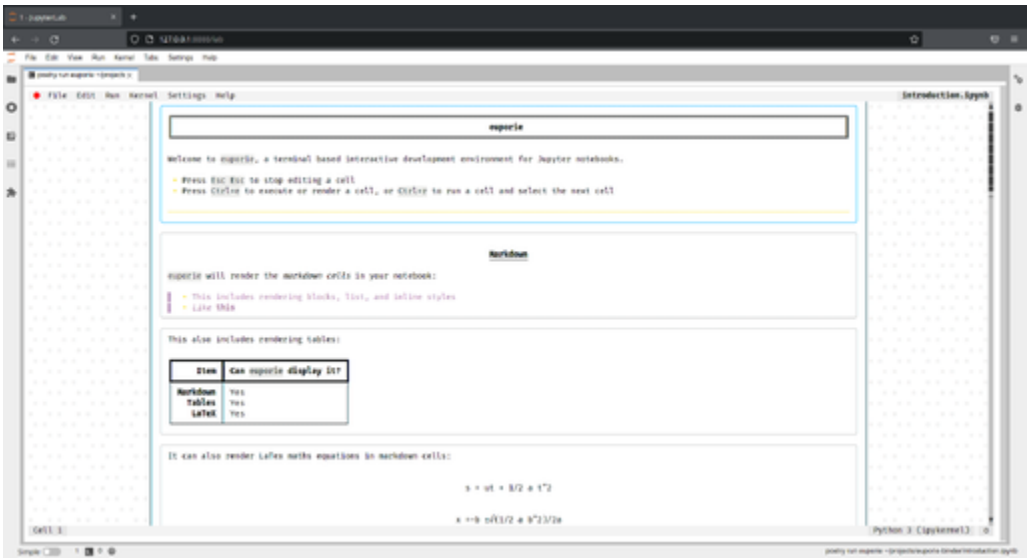


Fig. 19: Running inside JupyterLab

### 5.2.3 v0.x.x

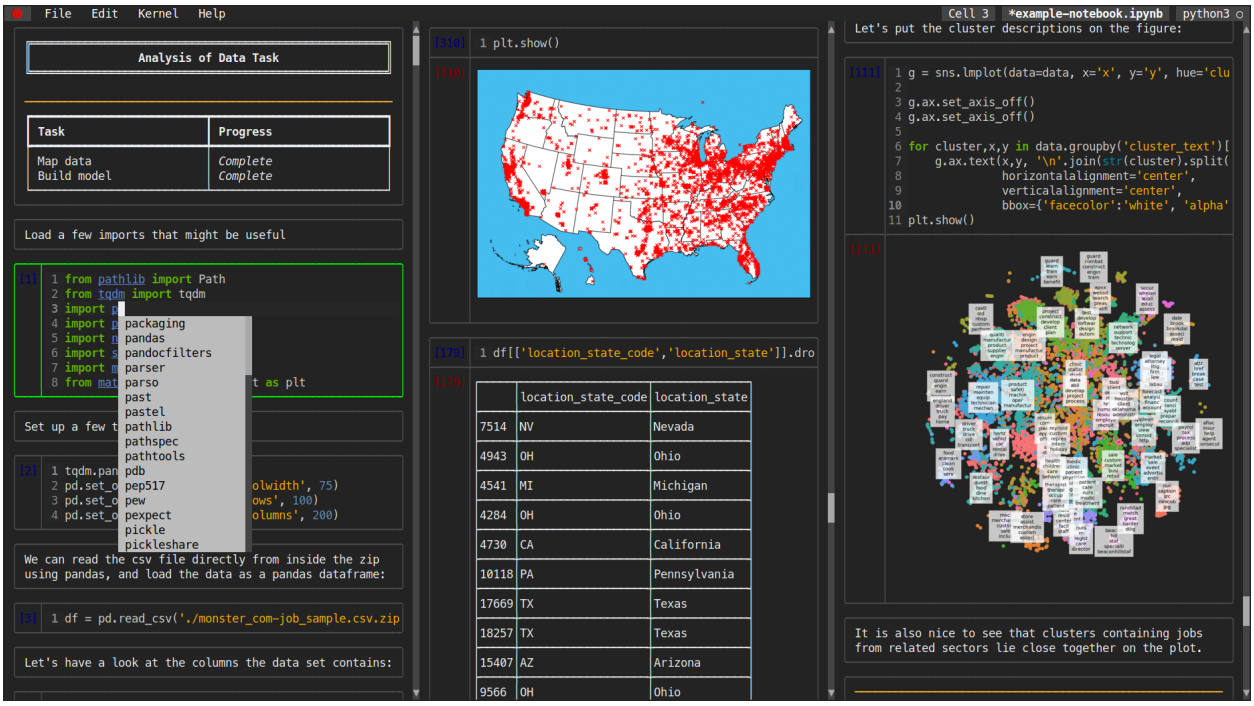



Fig. 20: Tiling notebooks vertically

## 5.3 Overview

Euporie consists of four applications, each providing different functionality:

***Euporie Notebook***
: This is an interactive TUI Jupyter Notebook editor, which runs in the terminal and allows you to view, edit and run Jupyter Notebooks. It makes full use of Jupyter's rich output system, and is able to render a range of media in the terminal.

***Euporie Console***
: This is a terminal frontend for kernels using the Jupyter protocol, which makes use of euporie's rich display system for kernel output. It is capable for displaying interactive widgets using `ipywidgets`. You can convert the history of a console session to a notebook.

***Euporie Preview***
: This allows you to preview notebooks in the terminal without opening them in the editor. The rendered notebook output can be printed to the standard output, piped to a pager, or saved to a file. You can also run notebooks before rendering them.

***Euporie Hub***
: This allows you to run euporie as a multi-user SSH server, meaning that multiple users can share a server's resources. Each connected user is presented with the euporie editor interface, and can use it to open, run and edit notebooks.

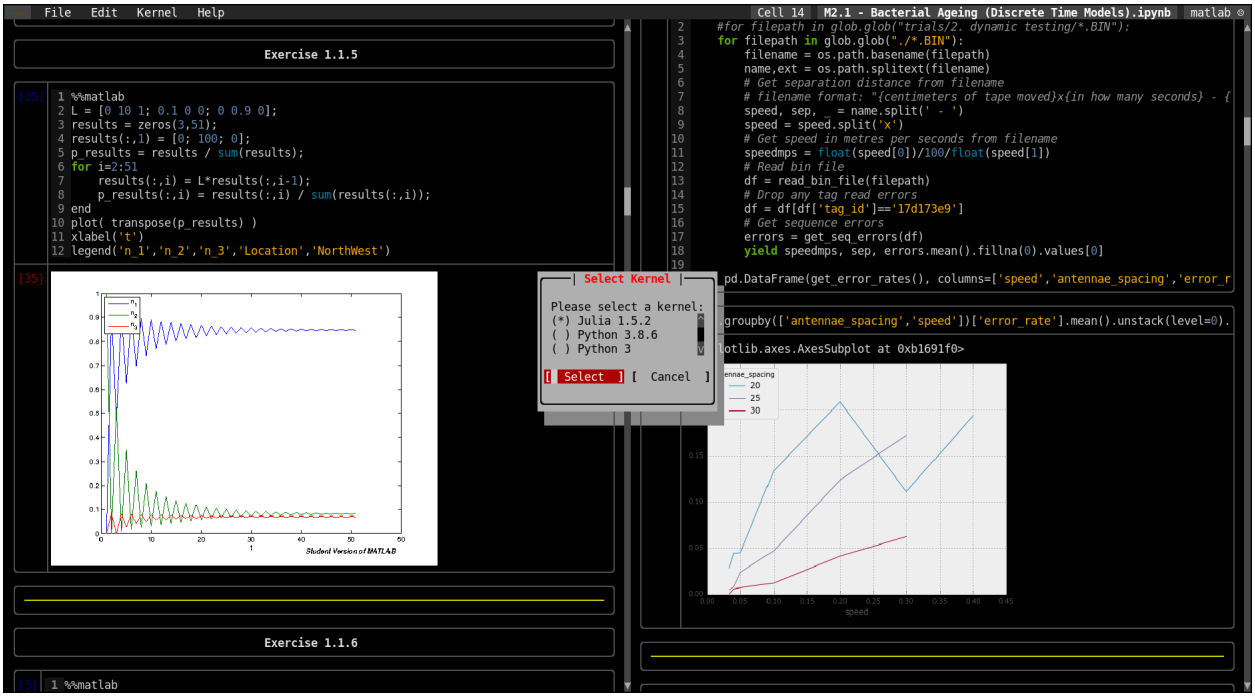



Fig. 21: Switching kernels

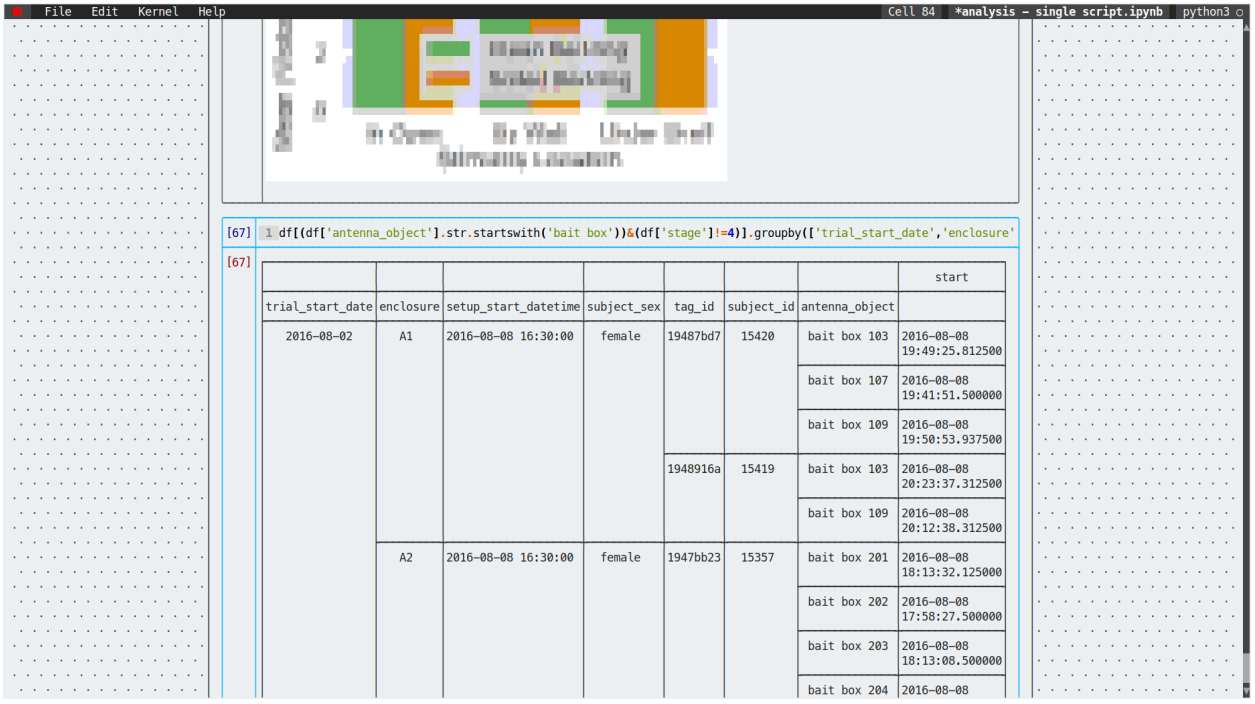



Fig. 22: Displaying multi-indexed dataframes

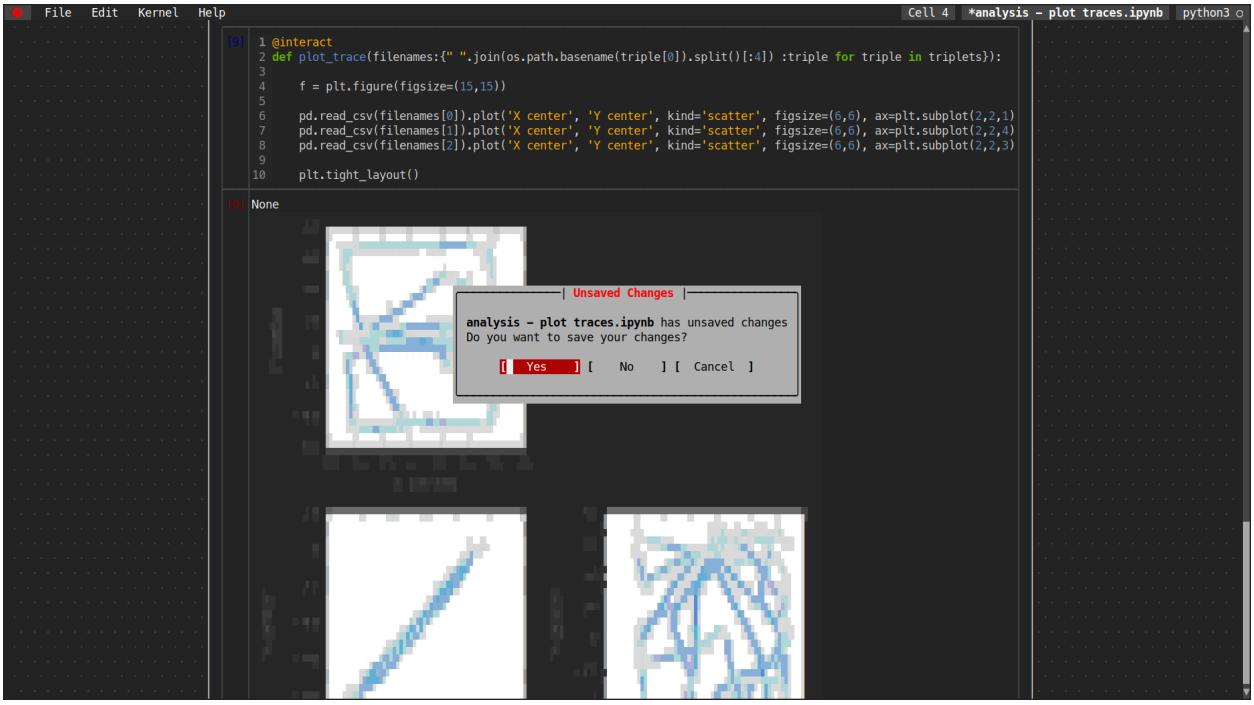



Fig. 23: Rendering images using unicode characters


Fig. 24: Running on a terminal with solarized light theme

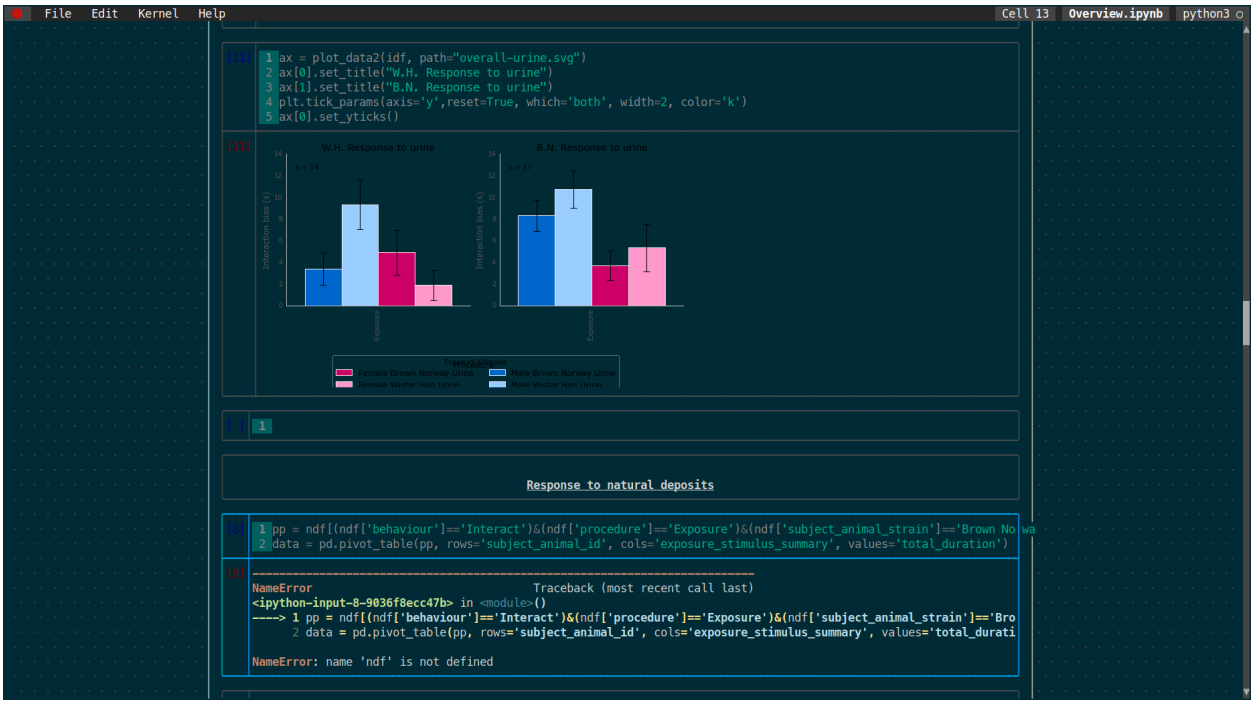



Fig. 25: Running on a terminal with solarized dark theme

### 5.3.1 Command Line Interface

In addition to being launched directly, each application in eurporie can be launched by passing its name as an argument to the **`euporie`** command.

#### Usage

```
$ euporie [-h] [--version] [--log-file [str]]
          [--log-level {debug,info,warning,error,critical}]
          [--log-config str]
          {console,edit,hub,notebook,preview}
```

### Positional Arguments

**{console,edit,hub,notebook,preview}**
The application to launch

### Optional Arguments

**-h, --help**
show this help message and exit

**--version, -V**
Show the version number and exit

**--log-file** <str>
File path for logs

**--log-level** {debug,info,warning,error,critical}
Set the log level

**--log-config** <str>
Additional logging configuration

## 5.4 Key Bindings

### 5.4.1 Editing Modes

The key-bindings used when editing a cell or in the console 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.

### 5.4.2 Custom Key Bindings

Key bindings can be customized by setting the `key_bindings` configuration parameter.

This parameter takes the form of a mapping, where they keys are references to modes or components to which a set of key-bindings apply, and the values are mapping of command names to lists of keys.

Key bindings set in the configuration will entirely over-ride the default binding, so if you want to add an additional binding for a command while retaining the default, you will need to include the default binding in the configuration.

Below is an example *Configuration File* showing how the key-bindings can be set:

```
{
  "notebook": {
    "autoformat": false,
    "expand": true,
    "key_bindings": {
      "euporie.notebook.app.NotebookApp": {
        "quit": ["c-q", "c-p"],
        "new-notebook": []
      }
    },
  }
}
```

This example sets two key-bindings in the *Notebook* app for the **`quit`** command: `Ctrl+Q` and `Ctrl+P`. It also unsets any key-bindings for the **`new-notebook`** command.

Custom key-binding configuration can also be passed on the command line in the form of a JSON string:

```
$ euporie-notebook --key-bindings='{"euporie.notebook.app.NotebookApp": {"new-notebook
↪": [],"quit": ["c-q", "c-p"]}}'
```

Valid component names include:

- `euporie.core.app.BaseApp`
- `euporie.core.widgets.pager.Pager`
- `euporie.core.widgets.inputs.KernelInput`
- `euporie.core.key_binding.bindings.micro.EditMod`
- `euporie.notebook.app.NotebookApp`
- `euporie.notebook.tabs.notebook.Notebook`
- `euporie.console.app.ConsoleApp`
- `euporie.console.tabs.console.Console`
- `euporie.preview.app.PreviewApp`

Most command names are listed in *Default Key Bindings Reference*.

### 5.4.3 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 `$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 |

## 5.4.4 Default Key Bindings Reference

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

### All euporie apps

| Keys | Description | Command |
|---|---|---|
| `Ctrl+Q` `<Sigint>` | Quit euporie. | **`quit`** |
| `Ctrl+W` | Close the current tab. | **`close-tab`** |
| `Ctrl+Page-down` | Switch to the next tab. | **`next-tab`** |
| `Ctrl+Pageup` | Switch to the previous tab. | **`previous-tab`** |
| `Shift+Tab` | Focus the next control. | **`focus-next`** |
| `Tab` `Ctrl+I` | Focus the previous control. | **`focus-previous`** |
| `Ctrl+L` | Clear the screen and the previous output. | **`clear-screen`** |
| `Tab` `Ctrl+I` | Show the completion menu and select the next completion. | **`next-completion`** |
| `Shift+Tab` | Show the completion menu and select the previous completion. | **`previous-completion`** |
| `Escape` | Cancel a completion. | **`cancel-completion`** |
| `Enter` `Ctrl+M` | Accept a selected completion. | **`accept-completion`** |
| `Ctrl+O` | Open a file. | **`open-file`** |
| `Alt+S` | Save the current file at a new location. | **`save-as`** |
| `Ctrl+Space` `Ctrl+@` | Show the command palette. | **`show-command-palette`** |
| `Ctrl+F` `F3` `F7` | Enter search mode. | **`find`** |
| `Ctrl+G` | Find the next search match. | **`find-next`** |
| `Ctrl+P` | Find the previous search match. | **`find-previous`** |

### Base class for interface tabs

| Keys | Description | Command |
|---|---|---|
| `Ctrl+S` | Save the current file. | **`save-file`** |
| `F5` | Reload the tab contents and reset the tab. | **`refresh-tab`** |

### Notebook app

| Keys | Description | Command |
|---|---|---|
| `Ctrl+B` | Open or close the current side-bar pane. | **`toggle-side-bar-pane`** |
| `Ctrl+N` | Create a new file. | **`new-notebook`** |
| `Alt+M` | Show the top bar | **`toggle-show-top-bar`** |

### Interactive notebooks

| Keys | Description | Command |
|---|---|---|
| `Enter`<br>`Ctrl+M` | Enter cell edit mode. | **enter-cell-edit-mode** |
| `Escape` | Exit cell edit mode. | **exit-edit-mode** |
| `Ctrl+Enter`<br>`Ctrl+E`<br>`Ctrl+Shift+F8` | Run or render the current cells. | **run-selected-cells** |
| `Shift+Enter`<br>`Ctrl+R`<br>`Shift+F9` | Run or render the current cells and select the next cell. | **run-and-select-next** |
| `Alt+Enter` | Run or render the current cells and insert a new cell below. | **run-cell-and-insert-below** |
| `A` | Add a new cell above the current. | **add-cell-above** |
| `B` | Add a new cell below the current. | **add-cell-below** |
| `D, D` | Delete the current cells. | **delete-cells** |
| `Z` | Undelete the last deleted cells. | **undelete-cells** |
| `X` | Cut the current cells. | **cut-cells** |
| `C` | Copy the current cells. | **copy-cells** |
| `Alt+C` | Copy the cell's output to the clipboard. | **copy-outputs** |
| `V` | Pate the previously copied cells. | **paste-cells** |
| `I, I` | Interrupt the notebook's kernel. | **interrupt-kernel** |
| `0, 0` | Restart the notebook's kernel. | **restart-kernel** |
| `[`<br>`<Scroll-Up>` | Scroll the page up a line. | **scroll-up** |
| `]`<br>`<Scroll-Down>` | Scroll the page down a line. | **scroll-down** |
| `{` | Scroll the page up 5 lines. | **scroll-up-5-lines** |
| `}` | Scroll the page down 5 lines. | **scroll-down-5-lines** |
| `Home`<br>`Ctrl+Up` | Select the first cell in the notebook. | **select-first-cell** |
| `Pageup` | Go up 5 cells. | **select-5th-previous-cell** |
| `Up`<br>`K` | Go up one cell. | **select-previous-cell** |
| `Down`<br>`J` | Select the next cell. | **select-next-cell** |
| `Pagedown` | Go down 5 cells. | **select-5th-next-cell** |
| `End`<br>`Ctrl+Down` | Select the last cell in the notebook. | **select-last-cell** |
| `Ctrl+A` | Select all cells in the notebook. | **select-all-cells** |
| `Shift+Home` | Extend the cell selection to the top of the notebook. | **extend-cell-selection-to-top** |
| `Shift+Up`<br>`Shift+K` | Extend the cell selection up a cell. | **extend-cell-selection-up** |
| `Shift+Down`<br>`Shift+J` | Extend the cell selection down a cell. | **extend-cell-selection-down** |
| `Shift+End` | Extend the cell selection to the bottom of the notebook. | **extend-cell-selection-to-bottom** |

continues on next page

Table 1 – continued from previous page

| Keys | Description | Command |
|---|---|---|
| `Alt+Up` | Move selected cells up. | **move-cells-up** |
| `Alt+Down` | Move selected cells down. | **move-cells-down** |
| `M` | Change selected cells to markdown cells. | **cells-to-markdown** |
| `Y` | Change selected cells to code cells. | **cells-to-code** |
| `R` | Change selected cells to raw cells. | **cells-to-raw** |
| `F` | Format the selected code cells. | **reformat-cells** |
| `Shift+F` | Automatically reformat all code cells in the notebook. | **reformat-notebook** |
| `E` | Edit cell in $EDITOR. | **edit-in-external-editor** |
| `Shift+M` | Merge the selected cells. | **merge-cells** |
| `Ctrl+\` | Split the current cell at the cursor position. | **split-cell** |
| `Up` | Move the cursor up to the previous cell. | **edit-previous-cell** |
| `Down` | Move the cursor down to the next cell. | **edit-next-cell** |
| `K` | Move the cursor up to the previous cell. | **edit-previous-cell-vi** |
| `J` | Move the cursor down to the next cell. | **edit-next-cell-vi** |
| `Left` | Scroll the cell output to the left. | **scroll-output-left** |
| `Right` | Scroll the cell output to the right. | **scroll-output-right** |
| `W` | Use the full width to display notebooks | **toggle-expand** |
| `Shift+W` | Wrap cell output text. | **toggle-wrap-cell-outputs** |
| `L` | Toggle line numbers when a buffer does not have focus. | **notebook-toggle-line-numbers** |

### Interactive console

| Keys | Description | Command |
|---|---|---|
| `Ctrl+C <Sigint>` | Clear the console input. | **clear-input** |
| `Ctrl+C <Sigint>` | Interrupt the notebook's kernel. | **cc-interrupt-kernel** |
| `Ctrl+Enter Ctrl+E` | Run the console input. | **run-input** |
| `Ctrl+D` | Signals the end of the input, causing the console to exit. | **end-of-file** |
| `Ctrl+L` | Clear the screen and the previous output. | **clear-screen** |

### Preview app

| Keys | Description | Command |
|---|---|---|
| `Ctrl+C`<br>`Ctrl+Q` | Quit euporie. | **`quit`** |

### Micro style editor key-bindings

| Keys | Description | Command |
|---|---|---|
| `Right` | Move forward a character, or down a line. | **`move-cursor-right`** |
| `Left` | Move back a character, or up a line. | **`move-cursor-left`** |
| `Enter`<br>`Ctrl+M` | Inert a new line, replacing any selection and indenting if appropriate. | **`newline`** |
| `Enter`<br>`Ctrl+M` | Accept an input. | **`accept-line`** |
| `Backspace`<br>`Ctrl+H`<br>`Backspace`<br>`Ctrl+H` | Delete the character behind the cursor. | **`backspace`** |
| `Ctrl+Backspace`<br>`Alt+Backspace`<br>`Ctrl+Alt+H`<br>`Alt+Ctrl+H` | Delete the word behind the cursor, using whitespace as a word boundary. | **`backward-kill-word`** |
| `Shift+Up`<br>`Shift+Down`<br>`Shift+Right`<br>`Shift+Left`<br>`Alt+Shift+Left`<br>`Alt+Shift+Right`<br>`Ctrl+Shift+Left`<br>`Ctrl+Shift+Right`<br>`Shift+Home`<br>`Shift+End`<br>`Ctrl+Shift+Home`<br>`Ctrl+Shift+End` | Start a new selection. | **`start-selection`** |
| `Shift+Up`<br>`Shift+Down`<br>`Shift+Right`<br>`Shift+Left`<br>`Alt+Shift+Left`<br>`Alt+Shift+Right`<br>`Ctrl+Shift+Left`<br>`Ctrl+Shift+Right`<br>`Shift+Home`<br>`Shift+End`<br>`Ctrl+Shift+Home`<br>`Ctrl+Shift+End` | Extend the selection. | **`extend-selection`** |

continues on next page

Table 2 – continued from previous page

| Keys | Description | Command |
|---|---|---|
| Up<br>Down<br>Right<br>Left<br>Alt+Left<br>Alt+Right<br>Ctrl+Left<br>Ctrl+Right<br>Home<br>End<br>Ctrl+Home<br>Ctrl+End | Cancel the selection. | **cancel-selection** |
| <Any> | Replace selection by what is typed. | **replace-selection** |
| Delete<br>Backspace<br>Ctrl+H<br>Backspace<br>Ctrl+H | Delete the contents of the current selection. | **delete-selection** |
| Ctrl+Left<br>Alt+B | Move back to the start of the current or previous word. | **backward-word** |
| Ctrl+Right<br>Alt+F | Move forward to the end of the next word. | **forward-word** |
| Alt+Up | Move the current or selected lines up by one line. | **move-lines-up** |
| Alt+Down | Move the current or selected lines down by one line. | **move-lines-down** |
| Home<br>Alt+Left<br>Alt+A | Move the cursor to the start of the line. | **go-to-start-of-line** |
| End<br>Alt+Right<br>Alt+E | Move the cursor to the end of the line. | **go-to-end-of-line** |
| Ctrl+Up<br>Ctrl+Home | Move to the start of the buffer. | **beginning-of-buffer** |
| Ctrl+Down<br>Ctrl+End | Move to the end of the buffer. | **end-of-buffer** |
| Alt+{ | Move the cursor to the start of the current paragraph. | **go-to-start-of-paragraph** |
| Alt+} | Move the cursor to the end of the current paragraph. | **go-to-end-of-paragraph** |
| Tab<br>Ctrl+I | Inndent the current or selected lines. | **indent-lines** |
| Backspace<br>Ctrl+H | Unindent the current or selected lines. | **unindent-line** |
| Shift+Tab | Unindent the current or selected lines. | **unindent-lines** |
| Ctrl+Z | Undo the last edit. | **undo** |
| Ctrl+Y | Redo the last edit. | **redo** |
| Ctrl+C | Add the current selection to the clipboard. | **copy-selection** |
| Ctrl+X<br>Shift+Delete | Remove the current selection and adds it to the clipboard. | **cut-selection** |
| Ctrl+K | Remove the current line adds it to the clipboard. | **cut-line** |
| Ctrl+D | Duplicate the current line. | **duplicate-line** |

continues on next page

Table 2 – continued from previous page

| Keys | Description | Command |
|---|---|---|
| Ctrl+D | Duplicate the current selection. | **duplicate-selection** |
| Ctrl+V | Pate the clipboard contents, replacing any current selection. | **paste-clipboard** |
| Ctrl+A | Select all text. | **select-all** |
| Delete | Delete character before the cursor. | **delete** |
| F4 | Toggle the case of the current word or selection. | **toggle-case** |
| Insert | Toggle overwrite when using micro editing mode. | **toggle-overwrite-mode** |
| Ctrl+U | Start recording a macro. | **start-macro** |
| Ctrl+U | Stop recording a macro. | **end-macro** |
| Ctrl+J | Re-execute the last keyboard macro defined. | **run-macro** |
| Right<br>Ctrl+F | Accept suggestion. | **accept-suggestion** |
| Alt+F | Fill partial suggestion. | **fill-suggestion** |
| Ctrl+/<br>Ctrl+_ | Comment or uncomments the current or selected lines. | **toggle-comment** |
| Alt+(<br>Alt+) | Go to matching bracket if the cursor is on a paired bracket. | **go-to-matching-bracket** |
| " | Wraps the current selection with: “” | **wrap-selection-""** |
| ' | Wraps the current selection with: ‘’ | **wrap-selection-''** |
| (<br>) | Wraps the current selection with: () | **wrap-selection-()** |
| {<br>} | Wraps the current selection with: {} | **wrap-selection-{}** |
| [<br>] | Wraps the current selection with: [] | **wrap-selection-[]** |
| ` | Wraps the current selection with: `` | **wrap-selection-``** |
| * | Wraps the current selection with: ** | **wrap-selection-**** |
| _ | Wraps the current selection with: __ | **wrap-selection-__** |

### Interactive help pager

| Keys | Description | Command |
|---|---|---|
| Escape<br>Q | Close the pager. | **close-pager** |

### Kernel input text areas

| Keys | Description | Command |
|---|---|---|
| Shift+Tab | Display contextual help. | **show-contextual-help** |
| Ctrl+Alt+Up | Get the previous history entry. | **history-prev** |
| Ctrl+Alt+Down | Get the next history entry. | **history-next** |

### Web view displays

| Keys | Description | Command |
|---|---|---|
| `Left` | Scroll the display up one line. | **scroll-web-view-left** |
| `Right` | Scroll the display down one line. | **scroll-web-view-right** |
| `Up`<br>`K` | Scroll the display up one line. | **scroll-webview-up** |
| `Down`<br>`J` | Scroll the display down one line. | **scroll-web-view-down** |
| `Pageup` | Scroll the display up one page. | **page-up-webview** |
| `Pagedown` | Scroll the display down one page. | **page-down-webview** |
| `Home` | Scroll the display to the top. | **go-to-start-of-webview** |
| `End` | Scroll the display down one page. | **go-to-end-of-web-view** |
| `Alt+Left` | Navigate backwards in the browser history. | **webview-nav-prev** |
| `Alt+Right` | Navigate forwards in the browser history. | **webview-nav-next** |

### Default Key-binding configuration

The following lists all of the default key-bindings used in euporie in the format required for custom key-bindings in the configuration file.

```
{
  "euporie.core.app.BaseApp": {
    "quit": [
      "c-q",
      "<sigint>"
    ],
    "close-tab": "c-w",
    "next-tab": "c-pagedown",
    "previous-tab": "c-pageup",
    "focus-next": "s-tab",
    "focus-previous": "tab",
    "clear-screen": "c-l",
    "next-completion": "c-i",
    "previous-completion": "s-tab",
    "cancel-completion": "escape",
    "accept-completion": "enter",
    "open-file": "c-o",
    "save-as": "A-s",
    "show-command-palette": "c-@",
    "find": [
      "c-f",
      "f3",
      "f7"
    ],
    "find-next": "c-g",
    "find-previous": "c-p"
```

(continues on next page)

(continued from previous page)

```
  },
  "euporie.core.tabs.base.Tab": {
    "save-file": "c-s",
    "refresh-tab": "f5"
  },
  "euporie.notebook.app.NotebookApp": {
    "toggle-side-bar-pane": "c-b",
    "new-notebook": "c-n",
    "toggle-show-top-bar": "A-m"
  },
  "euporie.notebook.tabs.notebook.Notebook": {
    "enter-cell-edit-mode": "enter",
    "exit-edit-mode": "escape",
    "run-selected-cells": [
      "c-enter",
      "c-e",
      "c-s-f8"
    ],
    "run-and-select-next": [
      "s-enter",
      "c-r",
      "s-f9"
    ],
    "run-cell-and-insert-below": "A-enter",
    "add-cell-above": "a",
    "add-cell-below": "b",
    "delete-cells": [
      "d",
      "d"
    ],
    "undelete-cells": "z",
    "cut-cells": "x",
    "copy-cells": "c",
    "copy-outputs": "A-c",
    "paste-cells": "v",
    "interrupt-kernel": [
      "i",
      "i"
    ],
    "restart-kernel": [
      "0",
      "0"
    ],
    "scroll-up": [
      "[",
      "<scroll-up>"
    ],
    "scroll-down": [
      "]",
      "<scroll-down>"
    ],
    "scroll-up-5-lines": "{",
    "scroll-down-5-lines": "}",
    "select-first-cell": [
      "home",
      "c-up"
    ],
```

(continues on next page)

(continued from previous page)

```
    "select-5th-previous-cell": "pageup",
    "select-previous-cell": [
      "up",
      "k"
    ],
    "select-next-cell": [
      "down",
      "j"
    ],
    "select-5th-next-cell": "pagedown",
    "select-last-cell": [
      "end",
      "c-down"
    ],
    "select-all-cells": "c-a",
    "extend-cell-selection-to-top": "s-home",
    "extend-cell-selection-up": [
      "s-up",
      "K"
    ],
    "extend-cell-selection-down": [
      "s-down",
      "J"
    ],
    "extend-cell-selection-to-bottom": "s-end",
    "move-cells-up": "A-up",
    "move-cells-down": "A-down",
    "cells-to-markdown": "m",
    "cells-to-code": "y",
    "cells-to-raw": "r",
    "reformat-cells": "f",
    "reformat-notebook": "F",
    "edit-in-external-editor": "e",
    "merge-cells": "M",
    "split-cell": "c-\\",
    "edit-previous-cell": "up",
    "edit-next-cell": "down",
    "edit-previous-cell-vi": "k",
    "edit-next-cell-vi": "j",
    "scroll-output-left": "left",
    "scroll-output-right": "right",
    "toggle-expand": "w",
    "toggle-wrap-cell-outputs": "W",
    "notebook-toggle-line-numbers": "l"
  },
  "euporie.console.app.ConsoleApp": {},
  "euporie.console.tabs.console.Console": {
    "clear-input": [
      "c-c",
      "<sigint>"
    ],
    "cc-interrupt-kernel": [
      "c-c",
      "<sigint>"
    ],
    "run-input": [
      "c-enter",
```

(continues on next page)

(continued from previous page)

```
      "c-e"
    ],
    "end-of-file": "c-d",
    "clear-screen": "c-l"
  },
  "euporie.preview.app.PreviewApp": {
    "quit": [
      "c-c",
      "c-q"
    ]
  },
  "euporie.core.key_binding.bindings.micro.EditMode": {
    "move-cursor-right": "right",
    "move-cursor-left": "left",
    "newline": "enter",
    "accept-line": "enter",
    "backspace": [
      "backspace",
      "c-h"
    ],
    "backward-kill-word": [
      "c-backspace",
      "A-backspace",
      "c-A-h",
      [
        "escape",
        "c-h"
      ]
    ],
    "start-selection": [
      "s-up",
      "s-down",
      "s-right",
      "s-left",
      "A-s-left",
      "A-s-right",
      "c-s-left",
      "c-s-right",
      "s-home",
      "s-end",
      "c-s-home",
      "c-s-end"
    ],
    "extend-selection": [
      "s-up",
      "s-down",
      "s-right",
      "s-left",
      "A-s-left",
      "A-s-right",
      "c-s-left",
      "c-s-right",
      "s-home",
      "s-end",
      "c-s-home",
      "c-s-end"
    ],
```

(continues on next page)

(continued from previous page)

```
    "cancel-selection": [
      "up",
      "down",
      "right",
      "left",
      "A-left",
      "A-right",
      "c-left",
      "c-right",
      "home",
      "end",
      "c-home",
      "c-end"
    ],
    "replace-selection": "<any>",
    "delete-selection": [
      "delete",
      "backspace",
      "c-h"
    ],
    "backward-word": [
      "c-left",
      "A-b"
    ],
    "forward-word": [
      "c-right",
      "A-f"
    ],
    "move-lines-up": "A-up",
    "move-lines-down": "A-down",
    "go-to-start-of-line": [
      "home",
      "A-left",
      "A-a"
    ],
    "go-to-end-of-line": [
      "end",
      "A-right",
      "A-e"
    ],
    "beginning-of-buffer": [
      "c-up",
      "c-home"
    ],
    "end-of-buffer": [
      "c-down",
      "c-end"
    ],
    "go-to-start-of-paragraph": "A-{",
    "go-to-end-of-paragraph": "A-}",
    "indent-lines": "tab",
    "unindent-line": "backspace",
    "unindent-lines": "s-tab",
    "undo": "c-z",
    "redo": "c-y",
    "copy-selection": "c-c",
    "cut-selection": [
```

(continues on next page)

(continued from previous page)

```
      "c-x",
      "s-delete"
    ],
    "cut-line": "c-k",
    "duplicate-line": "c-d",
    "duplicate-selection": "c-d",
    "paste-clipboard": "c-v",
    "select-all": "c-a",
    "delete": "delete",
    "toggle-case": "f4",
    "toggle-overwrite-mode": "insert",
    "start-macro": "c-u",
    "end-macro": "c-u",
    "run-macro": "c-j",
    "accept-suggestion": [
      "right",
      "c-f"
    ],
    "fill-suggestion": "A-f",
    "toggle-comment": "c-_",
    "go-to-matching-bracket": [
      "A-(",
      "A-)"
    ],
    "wrap-selection-\"\"": "\"",
    "wrap-selection-''": "'",
    "wrap-selection-()": [
      "(",
      ")"
    ],
    "wrap-selection-{}": [
      "{",
      "}"
    ],
    "wrap-selection-[]": [
      "[",
      "]"
    ],
    "wrap-selection-``": "`",
    "wrap-selection-**": "*",
    "wrap-selection-__": "_"
  },
  "euporie.core.widgets.pager.Pager": {
    "close-pager": [
      "escape",
      "q"
    ]
  },
  "euporie.core.widgets.inputs.KernelInput": {
    "show-contextual-help": "s-tab",
    "history-prev": "c-A-up",
    "history-next": "c-A-down"
  },
  "euporie.core.widgets.display.Display": {},
  "euporie.web.widgets.webview.WebViewControl": {
    "scroll-webview-left": "left",
    "scroll-webview-right": "right",
```

(continues on next page)

(continued from previous page)

```
    "scroll-webview-up": [
      "up",
      "k"
    ],
    "scroll-webview-down": [
      "down",
      "j"
    ],
    "page-up-webview": "pageup",
    "page-down-webview": "pagedown",
    "go-to-start-of-webview": "home",
    "go-to-end-of-webview": "end",
    "webview-nav-prev": "A-left",
    "webview-nav-next": "A-right"
  }
}
```

## 5.5 Configuration

### 5.5.1 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
```

Optionals set on the command line are not persisted, and only apply to the application which is being run.

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_` for global configuration settings.

To set a configuration option for an individual app, the environment variable should additionally be prefixed with the app's name: as `EUPORIE_NOTEBOOK_`, `EUPORIE_CONSOLE_`, `EUPORIE_PREVIEW_`, or `EUPORIE_HUB_` .

The global version of each configuration option's environment variable are listed in *Configuration Options*.

**Example**

```
$ EUPORIE_COLOR_SCHEME=light EUPORIE_SHOW_CELL_BORDERS=False EUPORIE_EXPAND=True␣
↪euporie-notebook 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*.

Settings can be applied to an individual application be specifying them under that application's name.

**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",
  "notebook": {
    "expand": false,
    "always_show_tab_bar": true,
    "show_cell_borders": false
  },
  "console": {
    "color_scheme": "default",
    "syntax_theme": "dracula"
  },
  "preview": {
    "show_cell_borders": 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.

### 5.5.2 Configuration Options

All available configuration options are listed below:

**`version`**

**flags**
`--version` or `-V`

**default**
`False`

**type**
`boolean`

**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

---

**`clipboard`**

**flags**
`--clipboard`

**environment variable**
`EUPORIE_CLIPBOARD`

**default**
`'external'`

**type**
`string`

**options**
`['external', 'internal', 'terminal']`

**description**
The preferred clipboard access method

The clipboard access method to use. - `external`: Data is saved to the system clipboard using OS native tooling. - `internal`: Clipboard data is only stored and usable inside euporie - it is

not saved to the system clipboard.

- **`terminal`: uses OSC52 escape sequences to retrieve and set the clipboard** contents. Requires your terminal emulator to support OSC52. Works over SSH.

**`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.

**`log_level`**

**flags**
`--log-level`

**environment variable**
`EUPORIE_LOG_LEVEL`

**default**
`'warning'`

**type**
`string`

**options**
`['debug', 'info', 'warning', 'error', 'critical']`

**description**
Set the log level

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

**`log_config`**

**flags**
`--log-config`

**environment variable**
`EUPORIE_LOG_CONFIG`

**type**
`string`

**description**
Additional logging configuration

A JSON string specifying additional logging configuration.

**`show_shadows`**

**flags**
`--show-shadows`

**environment variable**
`EUPORIE_SHOW_SHADOWS`

**default**
`True`

**type**
`boolean`

**description**
Show or hide shadows under menus and dialogs

Sets whether shadows are shown under dialogs and popup-menus.

**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.

**set_cursor_shape**

**flags**
`--set-cursor-shape`

**environment variable**
`EUPORIE_SET_CURSOR_SHAPE`

**default**
`True`

**type**
`boolean`

**description**
Whether to set the shape of the cursor depending on the editing mode

When set to True, the euporie will set the shape of the terminal's cursor to a beam in insert mode and and underline in replace mode when editing.

**cursor_blink**

**flags**
`--cursor-blink`

**environment variable**
`EUPORIE_CURSOR_BLINK`

**default**
`False`

**type**
`boolean`

**description**
Whether to blink the cursor

When set to True, the cursor will blink.

**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.

**`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 style 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.

**`terminal_polling_interval`**

**flags**
`--terminal-polling-interval`

**environment variable**
`EUPORIE_TERMINAL_POLLING_INTERVAL`

**default**
`0.0`

**type**
`number`

**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.

### `formatters`

**flags**
`--formatters`

**environment variable**
`EUPORIE_FORMATTERS`

**default**
`[]`

**type**
`array`

**description**
List of external code formatters

An array listing languages and commands of formatters to use for reformatting code cells. The command is an array of the command any any arguments. Code to be formatted is pass in via the standard input, and replaced with the standard output.

e.g.

```
[
    {“command”: [“ruff”, “format”, “-“], “languages”: [“python”]}, {“command”: [“black”, “-“], “languages”: [“python”]}, {“command”: [“isort”, “-“], “languages”: [“python”]}
]
```

### `syntax_theme`

**flags**
`--syntax-theme`

**environment variable**
`EUPORIE_SYNTAX_THEME`

**default**
`'euporie'`

**type**
`string`

**description**
Syntax highlighting theme

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

### `color_depth`

**flags**
`--color-depth`

**environment variable**
`EUPORIE_COLOR_DEPTH`

**type**
`integer`

**options**
[1, 4, 8, 24]

**description**
The color depth to use

The number of bits to use to represent colors displayable on the screen. If set to None, the supported color depth of the terminal will be detected automatically.

### multiplexer_passthrough

**flags**
--multiplexer-passthrough

**environment variable**
EUPORIE_MULTIPLEXER_PASSTHROUGH

**default**
False

**type**
boolean

**description**
Use passthrough from within terminal multiplexers

If set and euporie is running inside a terminal multiplexer (**screen** or **tmux**), then certain escape sequences will be passed-through the multiplexer directly to the terminal.

This affects things such as terminal color detection and graphics display.

for tmux, you will also need to ensure that allow-passthrough is set to on in your **tmux** configuration.

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

---

**Note:** As of version **tmux** version 3.4 sixel graphics are supported, which may result in better terminal graphics then using multiplexer passthrough.

---

### color_scheme

**flags**
--color-scheme

**environment variable**
EUPORIE_COLOR_SCHEME

**default**
'default'

**type**
string

**options**
['default', 'inverse', 'light', 'dark', 'black', 'white', 'custom']

**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.

**custom_background_color**

**flags**
`--custom-background-color` or `--custom-bg-color` or `--bg`

**environment variable**
`EUPORIE_CUSTOM_BACKGROUND_COLOR`

**default**
`'#073642'`

**type**
`string`

**description**
Background color for "Custom" color theme

The hex code of the color to use for the background in the "Custom" color scheme.

**custom_foreground_color**

**flags**
`--custom-foreground-color` or `--custom-fg-color` or `--fg`

**environment variable**
`EUPORIE_CUSTOM_FOREGROUND_COLOR`

**default**
`'#839496'`

**type**
`string`

**description**
Foreground color for "Custom" color theme

The hex code of the color to use for the foreground in the "Custom" color scheme.

**accent_color**

**flags**
`--accent-color`

**environment variable**
`EUPORIE_ACCENT_COLOR`

**default**
`'ansiblue'`

**type**
`string`

**description**
Accent color to use in the app

The hex code of a color to use for the accent color in the application.

**key_bindings**

**flags**
`--key-bindings`

**environment variable**
`EUPORIE_KEY_BINDINGS`

**default**
`{}`

**type**
`object`

**description**
Additional key binding definitions

A mapping of component names to mappings of command name to key-binding lists.

### `graphics`

**flags**
`--graphics`

**environment variable**
`EUPORIE_GRAPHICS`

**type**
`string`

**options**
`['none', 'sixel', 'kitty', 'iterm']`

**description**
The preferred graphics protocol

The graphics protocol to use, if supported by the terminal. If set to "none", terminal graphics will not be used.

### `force_graphics`

**flags**
`--force-graphics`

**environment variable**
`EUPORIE_FORCE_GRAPHICS`

**default**
`False`

**type**
`boolean`

**description**
Force use of specified graphics protocol

When set to `True`, the graphics protocol specified by the `graphics` configuration option will be used even if the terminal does not support it.

This is also useful if you want to use graphics in **`euporie-hub`**.

### `enable_language_servers`

**flags**
`--enable-language-servers` or `--lsp`

**environment variable**
`EUPORIE_ENABLE_LANGUAGE_SERVERS`

**default**
`False`

**type**
`boolean`

**description**
Enable language server support

When set to `True`, language servers will be used for liniting, code inspection, and code formatting.

Additional language servers can be added using the `language-servers` option.

**`language_servers`**

**flags**
`--language-servers`

**environment variable**
`EUPORIE_LANGUAGE_SERVERS`

**default**
`{}`

**type**
`object`

**description**
Language server configurations

Additional language servers can be defined here, e.g.:

{
"ruff": {"command": ["ruff-lsp"], "languages": ["python"]}, "pylsp": {"command": ["pylsp"], "languages": ["python"]}, "typos": {"command": ["typos-lsp"], "languages": []}
}

The following properties are required: - The name to be given to the the language server, must be unique - The command list consists of the process to launch, followed by any

command line arguments

- A list of language the language server supports. If no languages are

given, the language server will be used for documents of any language.

To disable one of the default language servers, its name can be set to an empty dictionary. For example, the following would disable the awk language server:

{
"awk-language-server": {},
}

**`wrap_cell_outputs`**

**flags**
`--wrap-cell-outputs`

**environment variable**
`EUPORIE_WRAP_CELL_OUTPUTS`

**default**
`False`

**type**
`boolean`

**description**
Wrap cell output text.

Whether text-based cell outputs should be wrapped.

**`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.

**`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.

**kernel_name**

**flags**
`--kernel-name` or `--kernel`

**environment variable**
`EUPORIE_KERNEL_NAME`

**default**
`'python3'`

**type**
`string`

**description**
The name of the kernel to start by default

The name of the kernel selected automatically by the console app or in new notebooks. If set to an empty string, the user will be asked which kernel to launch.

**record_cell_timing**

**flags**
`--record-cell-timing`

**environment variable**
`EUPORIE_RECORD_CELL_TIMING`

**default**
`False`

**type**
`boolean`

**description**
Should timing data be recorded in cell metadata.

When set, execution timing data will be recorded in cell metadata.

**max_stored_outputs**

**flags**
`--max-stored-outputs`

**environment variable**
`EUPORIE_MAX_STORED_OUTPUTS`

**default**
`100`

**type**
`integer`

**description**
The number of inputs / outputs to store in an in-memory notebook

Defines the maximum number of executed "cells" to store in case the console session is saved to a file or converted into a notebook.

**connection_file**

**flags**
`--connection-file` or `--kernel-connection-file`

**environment variable**
`EUPORIE_CONNECTION_FILE`

**type**
`string`

**description**
Attempt to connect to an existing kernel using a JSON connection info file

If the file does not exist, kernel connection information will be written to the file path provided.

If the file exists, kernel connection info will be read from the file, allowing euporie to connect to existing kernels.

**show_file_icons**

**flags**
`--show-file-icons`

**environment variable**
`EUPORIE_SHOW_FILE_ICONS`

**default**
`False`

**type**
`boolean`

**description**
Show file icons in the file manager

Whether file icons should be shown in the file manager.

These icons exist in the unicode private use area, and may require custom fonts such as `awesome-terminal-fonts` or `nerdfonts` to be installed.

**`mouse_support`**

**flags**
`--mouse-support`

**environment variable**
`EUPORIE_MOUSE_SUPPORT`

**type**
`boolean`

**description**
Enable or disable mouse support

When set to True, mouse support is enabled. When set to False, mouse support is disabled.

**`app`**

**environment variable**
`EUPORIE_APP`

**default**
`'notebook'`

**type**
`string`

**options**
`['console', 'edit', 'hub', 'notebook', 'preview']`

**description**
The application to launch

The name of the application to launch.

**`host`**

**flags**
`--host`

**environment variable**
`EUPORIE_HOST`

**default**
`''`

**type**
`string`

**description**
The host address to bind to

This determines the host address the euporie hub SSH server will bind to.

**`port`**

**flags**
`--port`

**environment variable**
`EUPORIE_PORT`

**default**
`8022`

**type**
`integer`

**description**
The port for the ssh server to use

This determines which port euporie will listen on for connections to euporie hub.

**`host_keys`**

**flags**
`--host-keys`

**environment variable**
`EUPORIE_HOST_KEYS`

**default**
`['/etc/ssh/ssh_host_ecdsa_key']`

**type**
`array`

**description**
Host keys to use for the SSH server

One or more SSH host key files to use for the euporie hub SSH server.

**`client_keys`**

**flags**
`--client-keys`

**environment variable**
`EUPORIE_CLIENT_KEYS`

**default**
`['~/.ssh/authorized_keys']`

**type**
`array`

**description**
Client public keys authorized to connect

One or more OpenSSH-style `authorized_keys` files, containing public keys for authorized clients.

**auth**

**flags**
`--auth`

**environment variable**
`EUPORIE_AUTH`

**default**
`True`

**type**
`boolean`

**description**
Allow unauthenticated access to euporie hub

When set, users will be able to access euporie hub without authentication.

> **Warning:** This option is dangerous, as arbitrary code can be executed through euporie apps.

**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.

**external_editor**

**flags**
`--external-editor`

**environment variable**
`EUPORIE_EXTERNAL_EDITOR`

**type**
`string`

**description**
Set the external editor to use.

A command to run when editing cells externally. The following strings in the command will be replaced with values which locate the cell being edited:

- `{top}`
- `{left}`
- `{bottom}`

- {right}
- {width}
- {height}

This is useful if you run euporie inside a tmux session, and wish to launch your editor in a pop-up pane. This can be achieved by setting this parameter to something like the following:

```
"tmux display-popup -x {left} -y {bottom} -w {width} -h {height} -B -E␣
↪micro"
```

**save_widget_state**

**flags**
--save-widget-state

**environment variable**
EUPORIE_SAVE_WIDGET_STATE

**default**
True

**type**
boolean

**description**
Save a notebook's widget state in the notebook metadata

When set to True, the state of any widgets in the current notebook will be saves in the notebook's metadata. This enables widgets to be displayed when the notebook is re-opened without having to re-run the notebook.

**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.

**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

**show_scroll_bar**

**flags**
--show-scroll-bar

**environment variable**
EUPORIE_SHOW_SCROLL_BAR

**default**
True

**type**
boolean

**description**
Show the scroll bar

Whether the scroll bar should be shown on the right of the screen.

**show_side_bar**

**flags**
--show-side-bar

**environment variable**
EUPORIE_SHOW_SIDE_BAR

**default**
False

**type**
boolean

**description**
Show the side-bar

Whether the side-bar should be shown at the side of the screen.

**tab_mode**

**flags**
--tab-mode

**environment variable**
EUPORIE_TAB_MODE

**default**
'stack'

**type**
string

**options**
['stack', 'tile_horizontally', 'tile_vertically']

**description**
The method used to display multiple tabs

Determines how multiple tabs are displayed when more than one tab is open. * `stack` displays one tab at a time with a tab-bar * `tile_horizontally` displays tabs side-by-side * `tile_vertically` displays tabs one-atop-the-next

**`always_show_tab_bar`**

**flags**
`--always-show-tab-bar`

**environment variable**
`EUPORIE_ALWAYS_SHOW_TAB_BAR`

**default**
`False`

**type**
`boolean`

**description**
Always show the tab bar

When set, the tab bar will always be shown - otherwise the tab bar is only shown when multiple tabs are open.

**`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.

Recommended characters include: “·”, “●”, “╳”, “╱”, “╲”, “▒”, “▓”, “▚”, “▞”, “╬”

**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*.

**run**

**flags**
`--run`

**environment variable**
`EUPORIE_RUN`

**default**
`False`

**type**
`boolean`

**description**
Run the notebook files when loaded

If set, notebooks will be run automatically when opened, or if previewing a file, the notebooks will be run before being output.

**show_top_bar**

**flags**
`--show-top-bar`

**environment variable**
`EUPORIE_SHOW_TOP_BAR`

**default**
`True`

**type**
`boolean`

**description**
Show the top bar

Whether the top bar should be shown at the top of the screen.

**save**

**flags**
`--save`

**environment variable**
`EUPORIE_SAVE`

**default**
`False`

**type**
`boolean`

**description**
Save the notebook after running it

If set, notebooks will be saved after they have been run. This setting only has any affect if the `run` setting is active.

**show_filenames**

**flags**
`--show-filenames`

**environment variable**
`EUPORIE_SHOW_FILENAMES`

**default**
`False`

**type**
`boolean`

**description**
Show the notebook filenames when previewing multiple notebooks

If set, the notebook filenames will be printed above each notebook's output when multiple notebooks are being previewed.

**cell_start**

**flags**
`--cell-start`

**environment variable**
`EUPORIE_CELL_START`

**type**
`integer`

**description**
The first cell to include in the preview

When set, only cells after the given cell index will be shown.

**cell_stop**

**flags**
`--cell-stop`

**environment variable**
`EUPORIE_CELL_STOP`

**type**
integer

**description**
The last cell to include in the preview

When set, only cells before the given cell index will be shown.

**output_file**

**flags**
--output-file

**environment variable**
EUPORIE_OUTPUT_FILE

**default**
'-'

**type**
string

**description**
Output path when previewing 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 previewing a notebook.

## 5.6 Changelog

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

### 5.6.1 v2.8.2 (2024-05-13)

#### Added

- Added `utftex` as a renderer for LaTeX math

#### Fixed

- Only patch *prompt_toolkit* when an app is launched, not at import
- Ensure all key-bindings are configurable (thanks @matheusfillipe)
- Fix rare bug causing exception when notebook has no cells
- Prevent race condition causing duplicate cells at startup
- Mark notebook as modified when changing cell type
- Prevent ruff formatter deleting buffer contents

---

### 5.6.2 v2.8.1 (2024-03-01)

#### Fixed

- Prevent pager mime selection issue
- Fix `Tab` key when numlock is enabled in kitty

---

### 5.6.3 v2.8.0 (2024-03-01)

#### Added

- Support Python 3.12
- Implement LSP server support
- Add new `--force-graphics` option, to allow using the graphics protocol even if not supported by the terminal
- Automatically disable mouse support on scroll-up in console to allow for terminal scrollback buffer scrolling. Mouse support is re-enabled on the next key-press.

#### Changed

- Euporie now can use any external code formatting tool instead of a limited ranger of Python code formatters

### Fixed

- Hide input overflow margin if line wrapping is turned on
- Do not load clipboard until app starts
- Make menu widget more a11y friendly by position cursor on selected menu-item
- Prompt to save dirty text files
- Prevent flickering in `euporie-hub` clients on server log output
- Notify of dead kernel immediately
- Handle non-existent file in the text file editor
- Fix issue with Select widget not scrolling beyond selection with scrollbar

---

## 5.6.4 v2.7.0 (2024-01-15)

### Added

- Implement CSS blink
- Implement progressive rendering in webview
- Pre-render terminal graphics to speed up notebook scrolling
- Add support for `ruff` code formatter
- Add support for inline LaTeX math in markdown
- Add `ziamath` LaTeX to SVG converter
- Set scroll offset to 1 on cell inputs

### Changed

- Rename `--tmux-graphics` option to `--multiplexer-passthrough`, and make it additionally work with GNU screen
- Improve `imagemagick` detection so it works on Debian

### Fixed

- Top line of app no longer disappears sometimes when opening file
- Redraw app on theme update
- Fix `PgUp` and `PgDn` keybindings in `ScrollingContainer`
- Fix issue with black code formatter when an unprintable character is entered
- Fix issue with image tearing when using `chafa.py` to render images
- Do not highlight brackets if a kernel input is not focused
- Use private color registers for sixel graphics
- Print the entirety of a scrolled input in euporie-console after running the input

- Greatly improved responsiveness when scrolling large cells
- Fix issue when moving cursor up between cells cursor, where cursor moved to second last character

---

### 5.6.5 v2.6.2 (2023-11-23)

#### Fixed

- Fix terminal graphics in euporie-console

---

### 5.6.6 v2.6.1 (2023-11-17)

#### Fixed

- Do not expand ipywidget HTML widgets (improves appearance of `tqdm.notebook` progress bars)
- Fix data update callbacks for HTML & Image ipywidgets
- Fix running multiple cells
- Prevent exception when extending cell selection to include last cell
- Prevent notebook app freezing when copying cell outputs

---

### 5.6.7 v2.6.0 (2023-11-13)

#### Added

- Allow scrolling to top of first cell and bottom of last cell with `up` and `down` if not visible
- Display inline images in markdown and HTML using terminal graphics

#### Fixed

- Kitty graphics now appear when using non-default color schemes
- Prevent disabled forms being focused
- Prevent rare error when closing a tab
- Prevent error dialog collapsing at small terminal sizes
- Correct error in escape code for querying terminal dimensions
- Fix cell output wrap toggle shortcut

---

### 5.6.8 v2.5.3 (2023-10-19)

#### Added

- Use *justify_content* for alignment in *flex* elements in HTML renderer
- Option to use OSC52 for clipboard

#### Fixed

- Ensure the color of drop-shadows gets updated if the color scheme is changed while the app is running

---

### 5.6.9 v2.5.2 (2023-10-14)

#### Fixed

- Fix graphic cropping in webview
- Prevent rare error on format conversion failure

---

### 5.6.10 v2.5.1 (2023-10-13)

#### Fixed

- Ensure extended key support is disabled at correct point in rendering process for apps running in alternate screen

---

### 5.6.11 v2.5.0 (2023-10-13)

#### Added

- Allow wrapping cell outputs
- Add support for `%load` and `%edit` magics
- Make icons in file browser configurable
- Implement `display: grid` support in HTML renderer
- Add terminal graphics support to webview
- Redirect kernel output to log

### Fixed

- Prevent entry of typed escape sequence codes into text areas
- Reset the terminal extended key mode at exit
- Limit horizontal scrolling of display areas
- Prevent error when commenting an empty cell
- Prevent moving through history in vi navigation mode
- Launch kernels from base prefix by default unless `ipykernel` is installed inside euporie's environment
- Use `.md` filename suffix when editing markdown cells in external editor
- Improve CSI-u escape sequence detection

### Changed

- Perform format conversions asynchronously
- Rename `reset-tab` command to `refresh-tab`

---

## 5.6.12 v2.4.3 (2023-06-07)

### Fixed

- Fix "Wrong color format" error when suggesting dictionary key completions

---

## 5.6.13 v2.4.2 (2023-06-05)

### Changed

- Add common SVG namespaces to HTML inline SVGs before conversion

### Fixed

- Force block graphic output from `viu`
- Fix calling asynchronous commands (e.g. converting console session to a notebook)

---

### 5.6.14 v2.4.1 (2023-05-25)

#### Fixed

- Fix dependency version conflict between `platformdirs` and `typing-extensions`

---

### 5.6.15 v2.4.0 (2023-05-24)

#### Changed

- Change from *appdirs* to *platformdirs* for resolving user configuration path
- Improve changes of successful format conversion by trying all conversion routes
- Improvements to responsiveness when opening files
- Update completion menu style
- Changed name of `hub` configuration item `` `no_auth `` to `auth` with inverse logic

#### Added

- Add web viewer tab for sufing the world wide web
- Add JSON viewer tab
- Add ability to open remote files from "open" dialog
- Add ability to select file open method
- Add support for opening scripts & markdown documents as notebooks using Jupytext
- Show full file-browser file path in status-bar
- Make all scrollbars clickable
- Add ability to select an existing kernel when changing a notebook's kernel
- Add "*.desktop" files
- Select text on find-next
- Add support for saving changes to text files

#### Fixed

- Fix pandas dataframe HTML output formatting with row multi-indices
- Prevent every `euporie-consle` run clearing the screen in `Konsole`
- Replace tabs with spaces in ANSI text output
- `SelectMultiple` widget is now styled consistently with other widgets
- Restore terminal state on unexpected exit signal
- Prevent `KeyError: 'log_file'` error when launching apps via `euporie` app
- Prevent unknown markdown code block language causing rendering error

- Fix unexpected cropped graphics when using `timg` with character aspects ≠ 0.5
- Make clickable scrollbars work correctly with Window containers
- Render cells when converted to markdown
- Prevent crash when connecting to `euporie-hub`
- Improve mime-type detection
- Make saving safer by saving to a temporary file first

---

### 5.6.16 v2.3.2 (2023-03-21)

#### Added

- Add `Ctrl+up` and `Ctrl+down` as key-bindings to move through kernel history in the console
- Allow closing notebook tabs with the middle mouse button
- Notify the user if the kernel dies unexpectedly

#### Fixed

- Re-enable display of large images in console
- Fix HTML table colspan border rendering issue
- Fix minor notebook scrolling issues
- Fix scrollbar dragging on tiled notebooks
- Prevent exception if kernel requests an unknown lexer
- Improve handling of kernel startup errors and reliability of changing kernels
- Rendering of LaTeX as terminal graphics

---

### 5.6.17 v2.3.1 (2023-02-05)

#### Added

- Warn about unrecognised configuration options in the log

#### Fixed

- Fix minor issue with `chafa.py` image renderer

### 5.6.18 v2.3.0 (2023-02-03)

#### Added

- Add `cahfa.py` image renderer
- Add command to clear screen
- Add a "Restart kernel and clear all output" command
- Add commands for clearing cell outputs
- The `scroll-up-5-lines` and `scroll-down-5-lines` commands now scroll to the top or bottom of the document if less than 5 lines remain
- "Open" and "Save As" dialogs now include a file browser
- Added a side-bar, which currently shows a file browser
- Add file display tab
- Minor usability improvements to widgets
- Partially obscured images are now rendered using terminal graphics

#### Fixed

- Change "toggle cell inputs" / "toggle cell outputs" click area to just prompt number
- Focus notebook page on click
- Add global dragging to slider widgets
- Fixed crash when merging last two cells in a notebook
- Relaxed dependency specification constraints

#### Changed

- Major re-write of HTML renderer (there is still work to be done on rendering the contents of inline elements).

---

### 5.6.19 v2.2.1 (2022-12-09)

#### Added

- Add a setting for graphics protocol preference

#### Fixed

- Fix PIL import error

---

### 5.6.20 v2.2.0 (2022-12-01)

#### Added

- Make drop-shadows configurable

#### Changed

- Changed to hatch for package build system

#### Fixed

- Fix automatic contextual help
- Second attempt to fix cell output left scroll issue

---

### 5.6.21 v2.1.5 (2022-12-01)

#### Fixed

- Attempt fix for cell output left scroll issue
- Prevent `background_tasks` error with `prompt_toolkit==3.0.30`
- Prevent error when adding a cell during initial render

---

### 5.6.22 v2.1.4 (2022-11-29)

#### Fixed

- Ensure all cells are re-rendered when a notebook tab is reset

---

### 5.6.23 v2.1.3 (2022-11-29)

#### Added

- New command to reset tabs, causing notebooks to be reloaded from the filesystem
- Improvements to the HTML renderer

#### Fixed

- Use valid defaults for foreground and background colors for applications
- Do not collapse cell when clicking on prompt itself, only the area below
- Ensure logo is visible in the documentation
- Fix code block language detection in HTML renderer

---

### 5.6.24 v2.1.2 (2022-11-14)

#### Fixed

- Prevent crash on Python 3.8

---

### 5.6.25 v2.1.1 (2022-10-31)

#### Fixed

- Ensure select uses available width in kernel selection dialog
- Do not cut when deleting word in micro edit mode

---

### 5.6.26 v2.1.0 (2022-10-29)

#### Added

- Make shadows under menus and dialogs semi-transparent
- Add ability to parse comma-separated CSS selectors to HTML parser
- Make dialogs draggable
- Allow configuring the external editor
- Add ability to inject cell position into external editor command (this allows and external editor to be run in a tmux popup)

#### Fixed

- Significant performance improvements when rendering HTML tables
- Close file after prompting to save a new file when euporie is closed
- Fix dialog button shortcut keys
- Display error messages in file open dialog
- Fixed broken mouse events in button widgets
- Prevent `IndexError` when deleting a selection of cells including the last cell

---

### 5.6.27 v2.0.9 (2022-10-26)

#### Added

- Do not show text selection when cell input is not focused
- Allow menus to be closed with `Escape` and opened with `F10`
- Enable cell inputs and outputs to be collapsed
- Make changing cursor shapes to showing editing mode configurable

#### Fixed

- Fixed issue where graphics are not displayed in console but are in notebook
- Re-apply style to cell input background
- Highlight selected trailing whitespace
- Reset button selection status on any global mouse event

---

### 5.6.28 v2.0.8 (2022-10-04)

#### Added

- Use SGR-pixel position for greater scrolling resolution when dragging scrollbar
- Clear selection in console before printing input
- Add ability to toggle top bar visibility
- Use cursor shapes to show the current input mode

#### Fixed

- Even more graphics rendering adjustments, including enabling sixel rendering with `chafa`
- Do not clear a cell's output when converting a cell to markdown

---

### 5.6.29 v2.0.7 (2022-08-31)

#### Fixed

- Fix various graphics rendering glitches
- Disable line wrapping before probing terminal to prevent unrecognised APCs moving the cursor to the next line

---

### 5.6.30 v2.0.6 (2022-08-30)

#### Fixed

- Prevent last cell of SIXEL images being overwritten
- Fix ubiqitous hyperlink issue
- Fix graphics detection for Konsole
- Hide kitty detection APC sequence in terminals which do not support APC codes

---

### 5.6.31 v2.0.5 (2022-08-29)

#### Added

- Add interactive JSON cell output preview

#### Fixed

- Sort configuration sub-menus
- Prevent crash when opening key-binding dialog
- Prevent jumping when scrolling if document is less than one page long
- Fixed issue with range sliders which caused a crash on notebook load

---

### 5.6.32 v2.0.4 (2022-08-28)

#### Added

- Add colorful command line help text

#### Changed

- Pre-render cells in background thread when notebook is loaded
- Make scrolling the notebook significantly less janky

#### Fixed

- Scroll the selected cell into view when entering cell edit mode
- Prevent excessive re-rendering of cells

---

### 5.6.33 v2.0.3 - (2022-08-26)

#### Fixed

- Handle deleted cells in cell mouse handle wrapper
- Fixed bug in `euporie-notebook` where cursor remains hidden when the app exits
- Fix inverted missing kernel logic

---

### 5.6.34 v2.0.2 - (2022-08-19)

#### Fixed

- Fixed bug where rendering LaTeX in HTML would sometime fail
- Fixed bug where ipywidget float log sliders would not accept intermediate values

---

### 5.6.35 v2.0.1 - (2022-08-18)

#### Fixed

- Fixed bug with rendering LaTeX maths in markdown cells

---

### 5.6.36 v2.0.0 - (2022-08-18)

#### Added

- Added new HTML renderer
- Added shortcut key hints in menus and buttons
- Added a shadow under the completions menu
- Allow changing color depth on the fly
- Key-bindings can now be edited in the configuration file
- Queue inputs which are run before kernel starts, and run them once the kernel has started
- Add ability to convert from console to notebook
- Set initial vi mode to navigation
- Add euporie console application
- Add "Save As…" command
- Allow opening file from remote file systems (e.g. `http:`, `hdfs:`, `gs:`, `s3:`, etc.)
- Add ability to undo deleting cells
- Add support for ipywidgets
- Allow changing app accent color
- Allow connecting to existing kernels with `kernel_connection_file` config option

#### Changed

- Auto-indent text on newline inside brackets in micro editor mode
- Improve quality of copied outputs
- Cells are now displayed as soon as they are rendered in `preview`
- Added the `--save` option in `preview`, which saves the notebook if `--run` is used
- Apps are now individually configurable
- The `edit` app is not called `notebook`
- The preferred method of launching apps is now using the `euporie-*` commands
- The style of the applications and widgets has been refresh
- Allow creating new notebooks without first passing a file path
- Use `fastjsonschema` to parse configuration more quickly
- Reduce memory usage by allowing the garbage collector to remove deleted cells and graphics
- Make clicking to select a cell pass the click event to the cell

#### Fixed

- Clicking on a cell to focus no longer results in a selection if the notebooks has to be scrolled
- Fix issue with hyperlinks taking over the screen
- Fix bug which prevented mouse scrolling in some circumstances
- Fix `--version` command line flag

---

### 5.6.37 v1.6.2 - (2022-05-09)

#### Changed

- Display multiple cursors in vi multi-cursor mode

---

### 5.6.38 v1.6.1 - (2022-05-08)

#### Changed

- Allow `file:` scheme links in markdown

#### Fixed

- Enable entering vi navigation mode
- Change "go to matching bracket" command key-binding in micro mode to `Alt+(` / `Alt+)`, so as not to conflict with the "find-next" command

---

### 5.6.39 v1.6.0 - (2022-04-26)

#### Added

- Add dialog explaining if no kernels are found
- Allow changing tabs by scrolling on them
- Add "Custom" color scheme, allowing foreground and background colours to be configured
- Add "Black" and "White" color schemes
- Add a tab bar, and tab stacking / tiling
- Detect terminal colors inside **`tmux`**

### Changed

- Fix bug where markdown was not rendered on some installs
- Update documentation
- Remove input flush timeout after escape key
- Use sub-command in the command line interface
- Styling changes
- Do not colour cell input box background if terminal background color is not detected
- Improve cell stdin focus logic

### Fixed

- Force cell input to be re-lexed when changing cell type
- Prevent OSC-8 link mis-detection bug by adding link IDs
- Fix regression making cell selection in long notebooks very slow

---

## 5.6.40 v1.5.0 - (2022-04-19)

### Added

- Allow extending selection by word using `Alt+Shift+Left` / `Alt+Shift+Right`
- Add euporie hub: a multi-client SSH server serving euporie
- Add search toolbar (searches cell input in edit mode)
- Use *prompt_toolkit* to format logging output
- Allow scrolling cell outputs with `left` and :kbd`right` in command mode
- Theme more elements based on current terminal theme

### Changed

- Optimize imports, reducing import times by ~50%
- Improve fix for missing first empty cells in `html2text` markdown tables

### Fixed

- Restore graphics in tmux functionality
- Allow entering edit mode by clicking on cell input
- Expand ~ in log file path
- Prevent IPython import race condition when IPython is installed
- Prevent down key moving to next cell if the cursor is on the last line of a cell when the completion menu is open

- Enable euporie to run on Window again
- Prevent an underscore being printed when the app is launched
- Only send terminal queries if the terminal supports it

---

### 5.6.41 v1.4.3 - (2022-03-30)

#### Added

- Notebook will scroll so the cursor is always visible when in edit mode
- Add ability to copy cell output
- Allow colour-depth to be manually configured
- Allow scroll bar visibility to be toggled

#### Fixed

- Ensure dumping notebooks works when stdin is not a TTY (this allows euporie to be used to preview notebooks in ranger)
- Prevent exception when navigating to an empty cell in edit mode
- Prevent wide tables from wrapping with `html2text` renderer

---

### 5.6.42 v1.4.2 - (2022-03-28)

#### Added

- Respond to kernel requests to clear cell outputs
- Used colored output with elinks HTML renderer

#### Fixed

- Fix parsing of html tables with empty first cells by `html2text`
- Fixed toggling comments if a line consists of a single right-stripped comment

#### Changed

- Parse environment variables a Python literals, so setting a binary variable to `"False"` no longer evaluates to `True`

---

### 5.6.43 v1.4.1 - (2022-03-26)

- Exception raised when attempting to strip an empty formatted text string which occasionally caused issues when rendering markdown

---

### 5.6.44 v1.4.0 - (2022-03-26)

#### Added

- Add `html2text` as a HTML renderer
- Add new markdown parser based on `markdown_it`
- Highlight matching brackets and add command (`Ctrl+g`) to jump between matching brackets
- Add shortcuts `Alt+Up` and `Alt+Down` to move cells
- Show notebook mode in status-bar
- Allow moving cursor between cells from first / last line
- Allow extended a notebooks' cell selection with the mouse (`Shift + Click` or `Ctrl + Click`)
- Add ability to extend cell selection to top or bottom of notebook

#### Changed

- Make cell edit mode persistent between cells

#### Fixed

- Only send terminal query escape codes if output is a TTY - this prevents disrupted output when dumping or paging notebooks
- Fix word selection in case toggle command
- Images now resized if terminal font size changes
- Fixed maultiple bugs with external image converters
- Fix editing cells in external editor

---

### 5.6.45 v1.3.1 - (2022-03-24)

#### Fixed

- Prevent `ssort` clearing cells with only comments

### 5.6.46 v1.3.1 - (2022-03-20)

#### Fixed

- Fix notebook dumping regression

---

### 5.6.47 v1.3.0 - (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

---

### 5.6.48 v1.2.2 - (2022-03-17)

#### Fixed

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

---

### 5.6.49 v1.2.1 - (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 is has loaded
- Fixed bug where nothing would be focused after a completion if pager was opened

---

## 5.6.50 v1.2.0 - (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

---

## 5.6.51 v1.1.0 - (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

---

## 5.6.52 v1.0.0 - (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 form 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 vertially 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

**Emacs IPython Notebook**
: A Jupyter Notebook client and integrated REPL in Emacs.

**nbterm**
: An alternative effort sponsored by QuantStack

**nbtermix**
: A fork of nbterm

**jpterm**
: A Jupyter HTTP client with a TUI using Textual (currently a work in progress)

### 5.7.2 Consoles

**jupyter-console**
: A terminal-based console frontend for Jupyter kernels.

### 5.7.3 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)

**nbview**
A lightweight utility for viewing Jupyter Notebook files in your terminal, written in Go

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

**jupytui**
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

**nbcat**
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

**nbcat**
A script to page Jupyter Notebooks in the terminal

## 5.8 Euporie Notebook

In euporie's notebooks editor, you can interactively edit and run Jupyter noteboks from the comfort of your terminal.

Euporie supports most of the features of Jupyter Notebooks, so users of Jupyter should find it intuitive to use.

### 5.8.1 User Guide

#### Creating a New Notebook

To create a new notebook from the command line, launch euporie with the desired new notebook file path as an argument:

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

To create a new notebook from within the TUI editor, select *File ▸ New* in the menus, or press the `Ctrl+N` keyboard shortcut.

#### Editing a Cell

The currently selected cell can be edited by pressing `Enter`, which will enter Edit Mode. This focuses the cell's input text area, and allows you to edit the cell's content.

To exit Edit Mode, you can press the `Esc` key when you are done editing, or you can run the cell.

#### Running a Cell

To run the currently selected cell, you can press `Ctrl+Enter` (providing your terminal *suports this key-binding*; you can also use `Ctrl+e`).

Alternatively, you can select *Run ▸ Run selected cells* in the menu.

#### Opening an Existing Notebook

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

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

To open a notebook from within the TUI editor, select *File ▸ Open* in the menus, or press the `Ctrl+O` keyboard shortcut. Enter the path of the notebook you wish to open into the file open dialog and press *OK*.

### Adding a new cell

You can press `b` to add a new cell below the currently selected cell. You can also press `a` to add a new cell above the currently selected cell.

### Navigating the Notebook

When a notebook is open, the currently selected cell is highlighted with a blue border. The selected cell can be changed using the `Up` and `Down` arrow keys (or `j` and `k`). Selecting a cell which is not currently visible will automatically scroll it into view.

You can select the first or last cell the the `Home` or `End` keys.

The notebook can be scrolled up and down one line at a time using the `[` and `]` keys respectively. Use the `{` and `}` keys to scroll up or down by 5 lines. You can also scroll the notebook with the mouse wheel, or by clicking on the notebook's scroll bar.

### Saving a Notebook

To save a notebook, press `Ctrl+S` or navigate to *File ▸ Save Notebook* in the menus.

### Closing a Notebook

To save a notebook, press `Ctrl+W` or navigate to *File ▸ Close File Notebook* in the menus.

### Changing a Cell's Type

To change the currently selected cells to:

- **code cells**, press `y`;
- **markdown cells**, press `m`;
- **raw cells**, press `r`.

### Restarting the Kernel

To restart the kernel, navigate to *Kernel ‣ Restart kernel* in the menus and select *Yes* in the confirmation dialog.

Alternatively, press the `0 0` keyboard shortcut.

### Changing the Kernel

To change a notebook's kernel, navigate to *Kernel ‣ Change kernel* in the menus.

### Setting the Color Scheme

To change the color scheme, navigate to *Settings ‣ Color Scheme* in the menus, then select the color scheme name you want to use.

To configure the custom color scheme, use the `custom_foreground_color` and `custom_background_color` configuration options.

### Setting the Syntax Theme

To change the syntax theme, navigate to *Settings ‣ Syntax Theme* in the menus, then select the syntax theme you want to use.

### Using the Command Palette

Practically every action in euporie (including all of those listed above) can be performed through the command palette, which can be summoned using `Ctrl+Space`. You can type in the input box to search for a command, use the arrow keys to select a command from the list, and press `Enter` to run the selected command.

### 5.8.2 Command Line Interface

#### Usage

```
$ euporie-notebook [-h] [--version]
                   [--clipboard {external,internal,terminal}]
                   [--log-file [str]]
                   [--log-level {debug,info,warning,error,critical}]
                   [--log-config str]
                   [--show-shadows | --no-show-shadows]
                   [--show-status-bar | --no-show-status-bar]
                   [--set-cursor-shape | --no-set-cursor-shape]
                   [--cursor-blink | --no-cursor-blink]
                   [--edit-mode {micro,emacs,vi}] [--tab-size int]
                   [--terminal-polling-interval float]
                   [--formatters loads] [--syntax-theme str]
                   [--color-depth {1,4,8,24}]
                   [--multiplexer-passthrough | --no-multiplexer-passthrough]
                   [--color-scheme {default,inverse,light,dark,black,white,
↪custom}]
                   [--custom-background-color str]
                   [--custom-foreground-color str] [--accent-color str]
                   [--key-bindings loads]
                   [--graphics {none,sixel,kitty,iterm}]
                   [--force-graphics | --no-force-graphics]
                   [--enable-language-servers | --lsp | --no-enable-language-
↪servers | --no-lsp]
                   [--language-servers loads]
                   [--wrap-cell-outputs | --no-wrap-cell-outputs]
                   [--line-numbers | --no-line-numbers]
                   [--autoformat | --no-autoformat]
                   [--autocomplete | --no-autocomplete]
                   [--autosuggest | --no-autosuggest]
                   [--autoinspect | --no-autoinspect] [--kernel-name str]
                   [--record-cell-timing | --no-record-cell-timing]
                   [--show-file-icons | --no-show-file-icons]
                   [--show-cell-borders | --no-show-cell-borders]
                   [--external-editor str]
                   [--save-widget-state | --no-save-widget-state]
                   [--max-notebook-width int] [--expand | --no-expand]
                   [--show-scroll-bar | --no-show-scroll-bar]
                   [--show-side-bar | --no-show-side-bar]
                   [--tab-mode {stack,tile_horizontally,tile_vertically}]
                   [--always-show-tab-bar | --no-always-show-tab-bar]
                   [--background-pattern {0,1,2,3,4,5}]
                   [--background-character str]
                   [--run-after-external-edit | --no-run-after-external-edit]
                   [--run | --no-run]
                   [--show-top-bar | --no-show-top-bar]
                   [UPath ...]
```

### Positional Arguments

**<UPath> ...**
List of file names to open

### Optional Arguments

**-h, --help**
show this help message and exit

**--version, -V**
Show the version number and exit

**--clipboard** {external,internal,terminal}
The preferred clipboard access method

**--log-file** <str>
File path for logs

**--log-level** {debug,info,warning,error,critical}
Set the log level

**--log-config** <str>
Additional logging configuration

**--show-shadows, --no-show-shadows**
Show or hide shadows under menus and dialogs

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

**--set-cursor-shape, --no-set-cursor-shape**
Whether to set the shape of the cursor depending on the editing mode

**--cursor-blink, --no-cursor-blink**
Whether to blink the cursor

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

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

**--terminal-polling-interval** <float>
Time between terminal colour queries

**--formatters** <loads>
List of external code formatters

**--syntax-theme** <str>
Syntax highlighting theme

**--color-depth** {1,4,8,24}
The color depth to use

**--multiplexer-passthrough, --no-multiplexer-passthrough**
Use passthrough from within terminal multiplexers

**--color-scheme** {default,inverse,light,dark,black,white,custom}
The color scheme to use

**--custom-background-color** <str>, **--custom-bg-color** <str>, **--bg** <str>
Background color for "Custom" color theme

**--custom-foreground-color** <str>, **--custom-fg-color** <str>, **--fg** <str>
Foreground color for "Custom" color theme

**--accent-color** <str>
Accent color to use in the app

**--key-bindings** <loads>
Additional key binding definitions

**--graphics** {none,sixel,kitty,iterm}
The preferred graphics protocol

**--force-graphics**, **--no-force-graphics**
Force use of specified graphics protocol

**--enable-language-servers**, **--lsp**, **--no-enable-language-servers**, **--no-lsp**
Enable language server support

**--language-servers** <loads>
Language server configurations

**--wrap-cell-outputs**, **--no-wrap-cell-outputs**
Wrap cell output text.

**--line-numbers**, **--no-line-numbers**
Show or hide line numbers

**--autoformat**, **--no-autoformat**
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

**--kernel-name** <str>, **--kernel** <str>
The name of the kernel to start by default

**--record-cell-timing**, **--no-record-cell-timing**
Should timing data be recorded in cell metadata.

**--show-file-icons**, **--no-show-file-icons**
Show file icons in the file manager

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

**--external-editor** <str>
Set the external editor to use.

**--save-widget-state, --no-save-widget-state**
Save a notebook's widget state in the notebook metadata

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

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

**--show-scroll-bar, --no-show-scroll-bar**
Show the scroll bar

**--show-side-bar, --no-show-side-bar**
Show the side-bar

**--tab-mode** {stack,tile_horizontally,tile_vertically}
The method used to display multiple tabs

**--always-show-tab-bar, --no-always-show-tab-bar**
Always show the tab bar

**--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

**--run-after-external-edit, --no-run-after-external-edit**
Run cells after editing externally

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

**--show-top-bar, --no-show-top-bar**
Show the top bar

---

### 5.8.3 Available Commands

#### Command Available in Euporie Notebook

**toggle-version**

**title**
Toggle version

**description**
Show the version number and exit

**switch-clipboard**

**title**
Switch clipboard

**description**
Switch the value of the "clipboard" configuration option.

**`set-clipboard-external`**

**title**
Set clipboard to external

**description**
Set the value of the “clipboard” configuration option to “external”

**`set-clipboard-internal`**

**title**
Set clipboard to internal

**description**
Set the value of the “clipboard” configuration option to “internal”

**`set-clipboard-terminal`**

**title**
Set clipboard to terminal

**description**
Set the value of the “clipboard” configuration option to “terminal”

**`switch-log-level`**

**title**
Switch the log level

**description**
Switch the value of the “log_level” configuration option.

**`set-log-level-debug`**

**title**
Set the log level to debug

**description**
Set the value of the “log_level” configuration option to “debug”

**`set-log-level-info`**

**title**
Set the log level to info

**description**
Set the value of the “log_level” configuration option to “info”

**`set-log-level-warning`**

**title**
Set the log level to warning

**description**
Set the value of the “log_level” configuration option to “warning”

**`set-log-level-error`**

**title**
Set the log level to error

**description**
Set the value of the “log_level” configuration option to “error”

**`set-log-level-critical`**

**title**
Set the log level to critical

**description**
Set the value of the “log_level” configuration option to “critical”

**`toggle-show-shadows`**

**title**
Toggle show shadows

**description**
Show or hide shadows under menus and dialogs

**`toggle-show-status-bar`**

**title**
Toggle status bar

**description**
Show the status bar

**`toggle-set-cursor-shape`**

**title**
Toggle set cursor shape

**description**
Whether to set the shape of the cursor depending on the editing mode

**`toggle-cursor-blink`**

**title**
Toggle cursor blink

**description**
Whether to blink the cursor

**`quit`**

**title**
Quit

**description**
Quit euporie.

**`close-tab`**

**title**
Close tab

**description**
Close the current tab.

**`next-tab`**

**title**
Next tab

**description**
Switch to the next tab.

**previous-tab**

**title**
Previous tab

**description**
Switch to the previous tab.

**focus-next**

**title**
Focus next

**description**
Focus the next control.

**focus-previous**

**title**
Focus previous

**description**
Focus the previous control.

**clear-screen**

**title**
Clear screen

**description**
Clear the screen.

**switch-edit-mode**

**title**
Switch Editor key bindings

**description**
Switch the value of the “edit_mode” configuration option.

**set-edit-mode-micro**

**title**
Set Editor key bindings to micro

**description**
Set the value of the “edit_mode” configuration option to “micro”

**set-edit-mode-emacs**

**title**
Set Editor key bindings to emacs

**description**
Set the value of the “edit_mode” configuration option to “emacs”

**set-edit-mode-vi**

**title**
Set Editor key bindings to vi

**description**
Set the value of the “edit_mode” configuration option to “vi”

**switch-tab-size**

**title**
Switch tab size

**description**
Switch the value of the "tab_size" configuration option.

**set-syntax-theme-abap**

**title**
Set syntax theme to abap

**description**
Set the value of the "syntax_theme" configuration option to "abap"

**set-syntax-theme-algol**

**title**
Set syntax theme to algol

**description**
Set the value of the "syntax_theme" configuration option to "algol"

**set-syntax-theme-algol_nu**

**title**
Set syntax theme to algol_nu

**description**
Set the value of the "syntax_theme" configuration option to "algol_nu"

**set-syntax-theme-arduino**

**title**
Set syntax theme to arduino

**description**
Set the value of the "syntax_theme" configuration option to "arduino"

**set-syntax-theme-autumn**

**title**
Set syntax theme to autumn

**description**
Set the value of the "syntax_theme" configuration option to "autumn"

**set-syntax-theme-bw**

**title**
Set syntax theme to bw

**description**
Set the value of the "syntax_theme" configuration option to "bw"

**set-syntax-theme-borland**

**title**
Set syntax theme to borland

**description**
Set the value of the "syntax_theme" configuration option to "borland"

**`set-syntax-theme-coffee`**

**title**
Set syntax theme to coffee

**description**
Set the value of the “syntax_theme” configuration option to “coffee”

**`set-syntax-theme-colorful`**

**title**
Set syntax theme to colorful

**description**
Set the value of the “syntax_theme” configuration option to “colorful”

**`set-syntax-theme-default`**

**title**
Set syntax theme to default

**description**
Set the value of the “syntax_theme” configuration option to “default”

**`set-syntax-theme-dracula`**

**title**
Set syntax theme to dracula

**description**
Set the value of the “syntax_theme” configuration option to “dracula”

**`set-syntax-theme-emacs`**

**title**
Set syntax theme to emacs

**description**
Set the value of the “syntax_theme” configuration option to “emacs”

**`set-syntax-theme-friendly_grayscale`**

**title**
Set syntax theme to friendly_grayscale

**description**
Set the value of the “syntax_theme” configuration option to “friendly_grayscale”

**`set-syntax-theme-friendly`**

**title**
Set syntax theme to friendly

**description**
Set the value of the “syntax_theme” configuration option to “friendly”

**`set-syntax-theme-fruity`**

**title**
Set syntax theme to fruity

**description**
Set the value of the “syntax_theme” configuration option to “fruity”

**`set-syntax-theme-github-dark`**

**title**
Set syntax theme to github-dark

**description**
Set the value of the “syntax_theme” configuration option to “github-dark”

**`set-syntax-theme-gruvbox-dark`**

**title**
Set syntax theme to gruvbox-dark

**description**
Set the value of the “syntax_theme” configuration option to “gruvbox-dark”

**`set-syntax-theme-gruvbox-light`**

**title**
Set syntax theme to gruvbox-light

**description**
Set the value of the “syntax_theme” configuration option to “gruvbox-light”

**`set-syntax-theme-igor`**

**title**
Set syntax theme to igor

**description**
Set the value of the “syntax_theme” configuration option to “igor”

**`set-syntax-theme-inkpot`**

**title**
Set syntax theme to inkpot

**description**
Set the value of the “syntax_theme” configuration option to “inkpot”

**`set-syntax-theme-lightbulb`**

**title**
Set syntax theme to lightbulb

**description**
Set the value of the “syntax_theme” configuration option to “lightbulb”

**`set-syntax-theme-lilypond`**

**title**
Set syntax theme to lilypond

**description**
Set the value of the “syntax_theme” configuration option to “lilypond”

**`set-syntax-theme-lovelace`**

**title**
Set syntax theme to lovelace

**description**
Set the value of the “syntax_theme” configuration option to “lovelace”

**set-syntax-theme-manni**

**title**
Set syntax theme to manni

**description**
Set the value of the “syntax_theme” configuration option to “manni”

**set-syntax-theme-material**

**title**
Set syntax theme to material

**description**
Set the value of the “syntax_theme” configuration option to “material”

**set-syntax-theme-monokai**

**title**
Set syntax theme to monokai

**description**
Set the value of the “syntax_theme” configuration option to “monokai”

**set-syntax-theme-murphy**

**title**
Set syntax theme to murphy

**description**
Set the value of the “syntax_theme” configuration option to “murphy”

**set-syntax-theme-native**

**title**
Set syntax theme to native

**description**
Set the value of the “syntax_theme” configuration option to “native”

**set-syntax-theme-nord-darker**

**title**
Set syntax theme to nord-darker

**description**
Set the value of the “syntax_theme” configuration option to “nord-darker”

**set-syntax-theme-nord**

**title**
Set syntax theme to nord

**description**
Set the value of the “syntax_theme” configuration option to “nord”

**set-syntax-theme-one-dark**

**title**
Set syntax theme to one-dark

**description**
Set the value of the “syntax_theme” configuration option to “one-dark”

**set-syntax-theme-paraiso-dark**

**title**
Set syntax theme to paraiso-dark

**description**
Set the value of the "syntax_theme" configuration option to "paraiso-dark"

**set-syntax-theme-paraiso-light**

**title**
Set syntax theme to paraiso-light

**description**
Set the value of the "syntax_theme" configuration option to "paraiso-light"

**set-syntax-theme-pastie**

**title**
Set syntax theme to pastie

**description**
Set the value of the "syntax_theme" configuration option to "pastie"

**set-syntax-theme-perldoc**

**title**
Set syntax theme to perldoc

**description**
Set the value of the "syntax_theme" configuration option to "perldoc"

**set-syntax-theme-rainbow_dash**

**title**
Set syntax theme to rainbow_dash

**description**
Set the value of the "syntax_theme" configuration option to "rainbow_dash"

**set-syntax-theme-rrt**

**title**
Set syntax theme to rrt

**description**
Set the value of the "syntax_theme" configuration option to "rrt"

**set-syntax-theme-sas**

**title**
Set syntax theme to sas

**description**
Set the value of the "syntax_theme" configuration option to "sas"

**set-syntax-theme-solarized-dark**

**title**
Set syntax theme to solarized-dark

**description**
Set the value of the "syntax_theme" configuration option to "solarized-dark"

**`set-syntax-theme-solarized-light`**

**title**
Set syntax theme to solarized-light

**description**
Set the value of the “syntax_theme” configuration option to “solarized-light”

**`set-syntax-theme-staroffice`**

**title**
Set syntax theme to staroffice

**description**
Set the value of the “syntax_theme” configuration option to “staroffice”

**`set-syntax-theme-stata-dark`**

**title**
Set syntax theme to stata-dark

**description**
Set the value of the “syntax_theme” configuration option to “stata-dark”

**`set-syntax-theme-stata-light`**

**title**
Set syntax theme to stata-light

**description**
Set the value of the “syntax_theme” configuration option to “stata-light”

**`set-syntax-theme-tango`**

**title**
Set syntax theme to tango

**description**
Set the value of the “syntax_theme” configuration option to “tango”

**`set-syntax-theme-trac`**

**title**
Set syntax theme to trac

**description**
Set the value of the “syntax_theme” configuration option to “trac”

**`set-syntax-theme-vim`**

**title**
Set syntax theme to vim

**description**
Set the value of the “syntax_theme” configuration option to “vim”

**`set-syntax-theme-vs`**

**title**
Set syntax theme to vs

**description**
Set the value of the “syntax_theme” configuration option to “vs”

**`set-syntax-theme-xcode`**

**title**
Set syntax theme to xcode

**description**
Set the value of the “syntax_theme” configuration option to “xcode”

**`set-syntax-theme-zenburn`**

**title**
Set syntax theme to zenburn

**description**
Set the value of the “syntax_theme” configuration option to “zenburn”

**`switch-color-depth`**

**title**
Switch color depth

**description**
Switch the value of the “color_depth” configuration option.

**`set-color-depth-1`**

**title**
Set color depth to 1

**description**
Set the value of the “color_depth” configuration option to “1”

**`set-color-depth-4`**

**title**
Set color depth to 4

**description**
Set the value of the “color_depth” configuration option to “4”

**`set-color-depth-8`**

**title**
Set color depth to 8

**description**
Set the value of the “color_depth” configuration option to “8”

**`set-color-depth-24`**

**title**
Set color depth to 24

**description**
Set the value of the “color_depth” configuration option to “24”

**`toggle-multiplexer-passthrough`**

**title**
Toggle multiplexer passthrough

**description**
Use passthrough from within terminal multiplexers

**`switch-color-scheme`**

**title**
Switch color scheme

**description**
Switch the value of the “color_scheme” configuration option.

**`set-color-scheme-default`**

**title**
Set color scheme to default

**description**
Set the value of the “color_scheme” configuration option to “default”

**`set-color-scheme-inverse`**

**title**
Set color scheme to inverse

**description**
Set the value of the “color_scheme” configuration option to “inverse”

**`set-color-scheme-light`**

**title**
Set color scheme to light

**description**
Set the value of the “color_scheme” configuration option to “light”

**`set-color-scheme-dark`**

**title**
Set color scheme to dark

**description**
Set the value of the “color_scheme” configuration option to “dark”

**`set-color-scheme-black`**

**title**
Set color scheme to black

**description**
Set the value of the “color_scheme” configuration option to “black”

**`set-color-scheme-white`**

**title**
Set color scheme to white

**description**
Set the value of the “color_scheme” configuration option to “white”

**`set-color-scheme-custom`**

**title**
Set color scheme to custom

**description**
Set the value of the “color_scheme” configuration option to “custom”

**`switch-graphics`**

**title**
Switch graphics

**description**
Switch the value of the “graphics” configuration option.

**`set-graphics-none`**

**title**
Set graphics to none

**description**
Set the value of the “graphics” configuration option to “none”

**`set-graphics-sixel`**

**title**
Set graphics to sixel

**description**
Set the value of the “graphics” configuration option to “sixel”

**`set-graphics-kitty`**

**title**
Set graphics to kitty

**description**
Set the value of the “graphics” configuration option to “kitty”

**`set-graphics-iterm`**

**title**
Set graphics to iterm

**description**
Set the value of the “graphics” configuration option to “iterm”

**`toggle-force-graphics`**

**title**
Toggle force graphics

**description**
Force use of specified graphics protocol

**`toggle-enable-language-servers`**

**title**
Toggle enable language servers

**description**
Enable language server support

**`type-key`**

**title**
Type key

**description**
Enter a key.

`next-completion`

**title**
Next completion

**description**
Show the completion menu and select the next completion.

`previous-completion`

**title**
Previous completion

**description**
Show the completion menu and select the previous completion.

`cancel-completion`

**title**
Cancel completion

**description**
Cancel a completion.

`accept-completion`

**title**
Accept completion

**description**
Accept a selected completion.

`toggle-overwrite-mode`

**title**
Toggle overwrite mode

**description**
Toggle overwrite when using micro editing mode.

`start-macro`

**title**
Start macro

**description**
Start recording a macro.

`end-macro`

**title**
End macro

**description**
Stop recording a macro.

`run-macro`

**title**
Run macro

**description**
Re-execute the last keyboard macro defined.

**`backspace`**

**title**
Delete previous character

**description**
Delete the character behind the cursor.

**`delete`**

**title**
Delete character

**description**
Delete character before the cursor.

**`backward-kill-word`**

**title**
Delete previous word

**description**
Delete the word behind the cursor, using whitespace as a word boundary.

**`backward-word`**

**title**
Move back one word

**description**
Move back to the start of the current or previous word.

**`forward-word`**

**title**
Move forward one word

**description**
Move forward to the end of the next word.

**`beginning-of-buffer`**

**title**
Move to the beginning of the input

**description**
Move to the start of the buffer.

**`end-of-buffer`**

**title**
Move to the end of the input

**description**
Move to the end of the buffer.

**`scroll-backward`**

**title**
Scroll backward

**description**
Scroll window up.

**scroll-forward**

**title**
Scroll forward

**description**
Scroll window down.

**scroll-half-page-down**

**title**
Scroll down half a page

**description**
Same as ControlF, but only scroll half a page.

**scroll-half-page-up**

**title**
Scroll up half a page

**description**
Same as ControlB, but only scroll half a page.

**scroll-one-line-down**

**title**
Scroll down one line

**description**
scroll_offset += 1.

**scroll-one-line-up**

**title**
Scroll up one line

**description**
scroll_offset -= 1.

**move-cursor-left**

**title**
Move cursor left

**description**
Move back a character, or up a line.

**move-cursor-right**

**title**
Move cursor right

**description**
Move forward a character, or down a line.

**go-to-start-of-line**

**title**
Go to start of line

**description**
Move the cursor to the start of the line.

**`go-to-end-of-line`**

**title**
Go to end of line

**description**
Move the cursor to the end of the line.

**`go-to-start-of-paragraph`**

**title**
Go to start of paragraph

**description**
Move the cursor to the start of the current paragraph.

**`go-to-end-of-paragraph`**

**title**
Go to end of paragraph

**description**
Move the cursor to the end of the current paragraph.

**`toggle-comment`**

**title**
Toggle comment

**description**
Comment or uncomments the current or selected lines.

**`wrap-selection-""`**

**title**
Wrap selection in “”

**description**
Wraps the current selection with: “”

**`wrap-selection-''`**

**title**
Wrap selection in ‘

**description**
Wraps the current selection with: ‘

**`wrap-selection-()`**

**title**
Wrap selection in ()

**description**
Wraps the current selection with: ()

**`wrap-selection-{}`**

**title**
Wrap selection in {}

**description**
Wraps the current selection with: {}

**wrap-selection-[]**

**title**
Wrap selection in []

**description**
Wraps the current selection with: []

**wrap-selection-``**

**title**
Wrap selection in ``

**description**
Wraps the current selection with: ``

**wrap-selection-****

**title**
Wrap selection in **

**description**
Wraps the current selection with: **

**wrap-selection-__**

**title**
Wrap selection in __

**description**
Wraps the current selection with: __

**wrap-selection-<>**

**title**
Wrap selection in <>

**description**
Wraps the current selection with: <>

**duplicate-line**

**title**
Duplicate line

**description**
Duplicate the current line.

**duplicate-selection**

**title**
Duplicate selection

**description**
Duplicate the current selection.

**paste-clipboard**

**title**
Paste

**description**
Pate the clipboard contents, replacing any current selection.

**`copy-selection`**

**title**
Copy

**description**
Add the current selection to the clipboard.

**`cut-selection`**

**title**
Cut

**description**
Remove the current selection and adds it to the clipboard.

**`cut-line`**

**title**
Cut line

**description**
Remove the current line adds it to the clipboard.

**`move-lines-up`**

**title**
Move lines up

**description**
Move the current or selected lines up by one line.

**`move-lines-down`**

**title**
Move lines down

**description**
Move the current or selected lines down by one line.

**`accept-line`**

**title**
Accept line

**description**
Accept an input.

**`newline`**

**title**
Newline

**description**
Inert a new line, replacing any selection and indenting if appropriate.

**`indent-lines`**

**title**
Indent lines

**description**
Inndent the current or selected lines.

`unindent-lines`

**title**
Unindent lines

**description**
Unindent the current or selected lines.

`unindent-line`

**title**
Unindent line

**description**
Unindent the current or selected lines.

`toggle-case`

**title**
Toggle case

**description**
Toggle the case of the current word or selection.

`undo`

**title**
Undo

**description**
Undo the last edit.

`redo`

**title**
Redo

**description**
Redo the last edit.

`select-all`

**title**
Select all

**description**
Select all text.

`start-selection`

**title**
Start selection

**description**
Start a new selection.

`extend-selection`

**title**
Extend selection

**description**
Extend the selection.

**`replace-selection`**

**title**
Replace selection

**description**
Replace selection by what is typed.

**`delete-selection`**

**title**
Delete selection

**description**
Delete the contents of the current selection.

**`cancel-selection`**

**title**
Cancel selection

**description**
Cancel the selection.

**`go-to-matching-bracket`**

**title**
Go to matching bracket

**description**
Go to matching bracket if the cursor is on a paired bracket.

**`accept-suggestion`**

**title**
Accept suggestion

**description**
Accept suggestion.

**`fill-suggestion`**

**title**
Fill suggestion

**description**
Fill partial suggestion.

**`scroll-page-down`**

**title**
Scroll page down

**description**
Scroll page down (prefer the cursor at the top of the page, after scrolling).

**`scroll-page-up`**

**title**
Scroll page up

**description**
Scroll page up (prefer the cursor at the bottom of the page, after scrolling).

**scroll-display-left**

**title**
Scroll display left

**description**
Scroll the display up one line.

**scroll-display-right**

**title**
Scroll display right

**description**
Scroll the display down one line.

**scroll-display-up**

**title**
Scroll display up

**description**
Scroll the display up one line.

**scroll-display-down**

**title**
Scroll display down

**description**
Scroll the display down one line.

**page-up-display**

**title**
Page up display

**description**
Scroll the display up one page.

**page-down-display**

**title**
Page down display

**description**
Scroll the display down one page.

**go-to-start-of-display**

**title**
Go to start of display

**description**
Scroll the display to the top.

**go-to-end-of-display**

**title**
Go to end of display

**description**
Scroll the display down one page.

**toggle-wrap-cell-outputs**

**title**
Toggle wrap cell outputs

**description**
Wrap cell output text.

**close-pager**

**title**
Close pager

**description**
Close the pager.

**toggle-line-numbers**

**title**
Toggle line numbers

**description**
Show or hide line numbers

**toggle-autoformat**

**title**
Toggle autoformat

**description**
Automatically re-format code cells when run

**toggle-autocomplete**

**title**
Toggle autocomplete

**description**
Provide completions suggestions automatically

**toggle-autosuggest**

**title**
Toggle autosuggest

**description**
Provide line completion suggestions

**toggle-autoinspect**

**title**
Toggle autoinspect

**description**
Display contextual help automatically

**show-contextual-help**

**title**
Show contextual help

**description**
Display contextual help.

**history-prev**

**title**
History prev

**description**
Get the previous history entry.

**history-next**

**title**
History next

**description**
Get the next history entry.

**reformat-input**

**title**
Reformat input

**description**
Format the contents of the current input field.

**refresh-tab**

**title**
Refresh the current tab

**description**
Reload the tab contents and reset the tab.

**reset-tab**

**title**
Reset the current tab

**description**
Reset the current tab

**save-file**

**title**
Save file

**description**
Save the current file.

**change-kernel**

**title**
Change kernel

**description**
Change the notebook's kernel.

**toggle-record-cell-timing**

**title**
Toggle cell timing recording

**description**
Should timing data be recorded in cell metadata.

**`toggle-show-file-icons`**

**title**
Toggle File icons

**description**
Show file icons in the file manager

**`about`**

**title**
About

**description**
Show the about dialog.

**`open-file`**

**title**
Open file

**description**
Open a file.

**`save-as`**

**title**
Save as

**description**
Save the current file at a new location.

**`keyboard-shortcuts`**

**title**
Keyboard shortcuts

**description**
Display details of registered key-bindings in a dialog.

**`toggle-command-palette`**

**title**
Toggle command palette

**description**
Show the command palette.

**`show-command-palette`**

**title**
Show command palette

**description**
Show the command palette.

**`hide-command-palette`**

**title**
Hide command palette

**description**
Hide the command palette.

`find`

**title**
Find

**description**
Enter search mode.

`find-next`

**title**
Find next

**description**
Find the next search match.

`find-previous`

**title**
Find previous

**description**
Find the previous search match.

`stop-search`

**title**
Stop search

**description**
Abort the search.

`accept-search`

**title**
Accept search

**description**
Accept the search input.

`view-logs`

**title**
View logs

**description**
Open the logs in a new tab.

`toggle-show-cell-borders`

**title**
Toggle cell borders

**description**
Show or hide cell borders.

`toggle-save-widget-state`

**title**
Toggle save widget state

**description**
Save a notebook's widget state in the notebook metadata

**`switch-max-notebook-width`**

**title**
Switch max notebook width

**description**
Switch the value of the "max_notebook_width" configuration option.

**`toggle-expand`**

**title**
Toggle expand

**description**
Use the full width to display notebooks

**`toggle-show-scroll-bar`**

**title**
Toggle scroll bar

**description**
Show the scroll bar

**`enter-cell-edit-mode`**

**title**
Enter cell edit mode

**description**
Enter cell edit mode.

**`exit-edit-mode`**

**title**
Exit edit mode

**description**
Exit cell edit mode.

**`run-selected-cells`**

**title**
Run selected cells

**description**
Run or render the current cells.

**`run-and-select-next`**

**title**
Run selected cells and select next cell

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

**`run-cell-and-insert-below`**

**title**
Run cell and insert below

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

`run-all-cells`

**title**
Run all cells

**description**
Run or render all the cells in the current notebook.

`add-cell-above`

**title**
Add cell above

**description**
Add a new cell above the current.

`add-cell-below`

**title**
Add cell below

**description**
Add a new cell below the current.

`delete-cells`

**title**
Delete cells

**description**
Delete the current cells.

`undelete-cells`

**title**
Undelete cells

**description**
Undelete the last deleted cells.

`cut-cells`

**title**
Cut cells

**description**
Cut the current cells.

`copy-cells`

**title**
Copy cells

**description**
Copy the current cells.

`copy-outputs`

**title**
Copy outputs

**description**
Copy the cell's output to the clipboard.

**paste-cells**

**title**
Paste cells

**description**
Pate the previously copied cells.

**merge-cells**

**title**
Merge cells

**description**
Merge the selected cells.

**scroll-up**

**title**
Scroll up

**description**
Scroll the page up a line.

**scroll-down**

**title**
Scroll down

**description**
Scroll the page down a line.

**scroll-up-5-lines**

**title**
Scroll up 5 lines

**description**
Scroll the page up 5 lines.

**scroll-down-5-lines**

**title**
Scroll down 5 lines

**description**
Scroll the page down 5 lines.

**select-first-cell**

**title**
Select first cell

**description**
Select the first cell in the notebook.

**select-5th-previous-cell**

**title**
Select 5th previous cell

**description**
Go up 5 cells.

`select-previous-cell`

**title**
Select previous cell

**description**
Go up one cell.

`select-next-cell`

**title**
Select next cell

**description**
Select the next cell.

`select-5th-next-cell`

**title**
Select 5th next cell

**description**
Go down 5 cells.

`select-last-cell`

**title**
Select last cell

**description**
Select the last cell in the notebook.

`select-all-cells`

**title**
Select all cells

**description**
Select all cells in the notebook.

`extend-cell-selection-to-top`

**title**
Extend cell selection to top

**description**
Extend the cell selection to the top of the notebook.

`extend-cell-selection-up`

**title**
Extend cell selection up

**description**
Extend the cell selection up a cell.

`extend-cell-selection-down`

**title**
Extend cell selection down

**description**
Extend the cell selection down a cell.

**`extend-cell-selection-to-bottom`**

**title**
Extend cell selection to bottom

**description**
Extend the cell selection to the bottom of the notebook.

**`move-cells-up`**

**title**
Move cells up

**description**
Move selected cells up.

**`move-cells-down`**

**title**
Move cells down

**description**
Move selected cells down.

**`cells-to-markdown`**

**title**
Cells to markdown

**description**
Change selected cells to markdown cells.

**`cells-to-code`**

**title**
Cells to code

**description**
Change selected cells to code cells.

**`cells-to-raw`**

**title**
Cells to raw

**description**
Change selected cells to raw cells.

**`clear-cell-outputs`**

**title**
Clear cell outputs

**description**
Clear the outputs of the selected cells.

**`clear-all-outputs`**

**title**
Clear all outputs

**description**
Clear the outputs of the selected cells.

**`show-cell-inputs`**

**title**
Expand cell inputs

**description**
Expand the selected cells' inputs.

**`hide-cell-inputs`**

**title**
Collapse cell inputs

**description**
Collapse the selected cells' inputs.

**`toggle-cell-inputs`**

**title**
Toggle cell inputs

**description**
Toggle the visibility of the selected cells' inputs.

**`show-cell-outputs`**

**title**
Expand cell outputs

**description**
Expand the selected cells' outputs.

**`hide-cell-outputs`**

**title**
Collapse cell outputs

**description**
Collapse the selected cells' outputs.

**`toggle-cell-outputs`**

**title**
Toggle cell outputs

**description**
Toggle the visibility of the selected cells' outputs.

**`reformat-cells`**

**title**
Reformat cells

**description**
Format the selected code cells.

**`reformat-notebook`**

**title**
Reformat notebook

**description**
Automatically reformat all code cells in the notebook.

**`edit-in-external-editor`**

**title**
Edit in external editor

**description**
Edit cell in $EDITOR.

**`split-cell`**

**title**
Split cell

**description**
Split the current cell at the cursor position.

**`edit-previous-cell`**

**title**
Edit previous cell

**description**
Move the cursor up to the previous cell.

**`edit-previous-cell-vi`**

**title**
Edit previous cell vi

**description**
Move the cursor up to the previous cell.

**`edit-next-cell`**

**title**
Edit next cell

**description**
Move the cursor down to the next cell.

**`edit-next-cell-vi`**

**title**
Edit next cell vi

**description**
Move the cursor down to the next cell.

**`scroll-output-left`**

**title**
Scroll output left

**description**
Scroll the cell output to the left.

**`scroll-output-right`**

**title**
Scroll output right

**description**
Scroll the cell output to the right.

`interrupt-kernel`

**title**
Interrupt kernel

**description**
Interrupt the notebook's kernel.

`restart-kernel`

**title**
Restart kernel

**description**
Restart the notebook's kernel.

`restart-kernel-and-clear-all-outputs`

**title**
Restart kernel and clear all outputs

**description**
Restart the notebook's kernel and clear all cell output.

`notebook-toggle-line-numbers`

**title**
Notebook toggle line numbers

**description**
Toggle line numbers when a buffer does not have focus.

`webview-nav-prev`

**title**
Webview nav prev

**description**
Navigate backwards in the browser history.

`webview-nav-next`

**title**
Webview nav next

**description**
Navigate forwards in the browser history.

`scroll-webview-left`

**title**
Scroll webview left

**description**
Scroll the display up one line.

`scroll-webview-right`

**title**
Scroll webview right

**description**
Scroll the display down one line.

**`scroll-webview-up`**

**title**
Scroll webview up

**description**
Scroll the display up one line.

**`scroll-webview-down`**

**title**
Scroll webview down

**description**
Scroll the display down one line.

**`page-up-webview`**

**title**
Page up webview

**description**
Scroll the display up one page.

**`page-down-webview`**

**title**
Page down webview

**description**
Scroll the display down one page.

**`go-to-start-of-webview`**

**title**
Go to start of webview

**description**
Scroll the display to the top.

**`go-to-end-of-webview`**

**title**
Go to end of webview

**description**
Scroll the display down one page.

**`toggle-show-side-bar`**

**title**
Toggle side-bar

**description**
Show the side-bar

**`toggle-side-bar-pane`**

**title**
Toggle side bar pane

**description**
Open or close the current side-bar pane.

**new-notebook**

**title**
New notebook

**description**
Create a new file.

**view-documentation**

**title**
View documentation

**description**
Open the documentation in a web-view tab.

**switch-tab-mode**

**title**
Switch tab mode

**description**
Switch the value of the “tab_mode” configuration option.

**set-tab-mode-stack**

**title**
Set tab mode to stack

**description**
Set the value of the “tab_mode” configuration option to “stack”

**set-tab-mode-tile_horizontally**

**title**
Set tab mode to tile_horizontally

**description**
Set the value of the “tab_mode” configuration option to “tile_horizontally”

**set-tab-mode-tile_vertically**

**title**
Set tab mode to tile_vertically

**description**
Set the value of the “tab_mode” configuration option to “tile_vertically”

**toggle-always-show-tab-bar**

**title**
Toggle always show tab bar

**description**
Always show the tab bar

**switch-background-pattern**

**title**
Switch background pattern

**description**
Switch the value of the “background_pattern” configuration option.

**set-background-pattern-0**

**title**
Set background pattern to 0

**description**
Set the value of the "background_pattern" configuration option to "0"

**set-background-pattern-1**

**title**
Set background pattern to 1

**description**
Set the value of the "background_pattern" configuration option to "1"

**set-background-pattern-2**

**title**
Set background pattern to 2

**description**
Set the value of the "background_pattern" configuration option to "2"

**set-background-pattern-3**

**title**
Set background pattern to 3

**description**
Set the value of the "background_pattern" configuration option to "3"

**set-background-pattern-4**

**title**
Set background pattern to 4

**description**
Set the value of the "background_pattern" configuration option to "4"

**set-background-pattern-5**

**title**
Set background pattern to 5

**description**
Set the value of the "background_pattern" configuration option to "5"

**toggle-run-after-external-edit**

**title**
Toggle run after external edit

**description**
Run cells after editing externally

**toggle-run**

**title**
Toggle run

**description**
Run the notebook files when loaded

**toggle-show-top-bar**

**title**
Toggle top bar

**description**
Show the top bar

## 5.9 Euporie Console

Euporie console is a terminal frontend for Jupyter kernels, allowing code to be executed interactively and rich output to be displayed.

Euporie console makes use of terminal graphics capabilities to display images, and will render markdown, LaTeX, and `ipywidgets`.

The command palette can be summoned with `Ctrl+Space`, which provides access to settings and various operations from within the console.

### 5.9.1 User Guide

#### Multi-line editing

The console will check the current input for completeness when the cursor is at the end of the input and `Enter` is pressed. If the input is determined to be incomplete, a new line will be inserted. Otherwise, the input will be executed.

If two blank lines are entered at the end of the input, the input will be executed.

A new-line can be inserted without checking the input for completeness using `Shift+Enter`.

The input can be executed immediately using `Ctrl+Enter` (providing your terminal *suports this key-binding*; you can also use `Ctrl+e`).

#### Save console session as a notebook

Running the `save-as` command from the command palette will prompt you to save the input and output history of the current console session as a Jupyter notebook file.

#### Convert console session to a notebook

Running the `convert-to-notebook` command from the command palette will transform the input and output history of the current console session into a euporie notebook, and open it with *Euporie Notebook*, reusing the existing kernel connection.

### 5.9.2 Command Line Interface

#### Usage

```
$ euporie-console [-h] [--version]
                  [--clipboard {external,internal,terminal}]
                  [--log-file [str]]
                  [--log-level {debug,info,warning,error,critical}]
                  [--log-config str] [--show-shadows | --no-show-shadows]
                  [--show-status-bar | --no-show-status-bar]
                  [--set-cursor-shape | --no-set-cursor-shape]
                  [--cursor-blink | --no-cursor-blink]
                  [--edit-mode {micro,emacs,vi}] [--tab-size int]
                  [--terminal-polling-interval float]
                  [--formatters loads] [--syntax-theme str]
                  [--color-depth {1,4,8,24}]
                  [--multiplexer-passthrough | --no-multiplexer-passthrough]
                  [--color-scheme {default,inverse,light,dark,black,white,custom}
↪]
                  [--custom-background-color str]
                  [--custom-foreground-color str] [--accent-color str]
                  [--key-bindings loads]
                  [--graphics {none,sixel,kitty,iterm}]
                  [--force-graphics | --no-force-graphics]
                  [--enable-language-servers | --lsp | --no-enable-language-
↪servers | --no-lsp]
                  [--language-servers loads]
                  [--wrap-cell-outputs | --no-wrap-cell-outputs]
                  [--line-numbers | --no-line-numbers]
                  [--autoformat | --no-autoformat]
                  [--autocomplete | --no-autocomplete]
                  [--autosuggest | --no-autosuggest]
                  [--autoinspect | --no-autoinspect] [--kernel-name str]
                  [--record-cell-timing | --no-record-cell-timing]
                  [--max-stored-outputs int] [--connection-file UPath]
                  [--show-file-icons | --no-show-file-icons]
                  [--mouse-support | --no-mouse-support]
                  [UPath ...]
```

#### Positional Arguments

**<UPath> ...**
: List of file names to open

#### Optional Arguments

**-h, --help**
: show this help message and exit

**--version, -V**
: Show the version number and exit

**--clipboard** {external,internal,terminal}
: The preferred clipboard access method

**--log-file** <str>
File path for logs

**--log-level** {debug,info,warning,error,critical}
Set the log level

**--log-config** <str>
Additional logging configuration

**--show-shadows, --no-show-shadows**
Show or hide shadows under menus and dialogs

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

**--set-cursor-shape, --no-set-cursor-shape**
Whether to set the shape of the cursor depending on the editing mode

**--cursor-blink, --no-cursor-blink**
Whether to blink the cursor

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

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

**--terminal-polling-interval** <float>
Time between terminal colour queries

**--formatters** <loads>
List of external code formatters

**--syntax-theme** <str>
Syntax highlighting theme

**--color-depth** {1,4,8,24}
The color depth to use

**--multiplexer-passthrough, --no-multiplexer-passthrough**
Use passthrough from within terminal multiplexers

**--color-scheme** {default,inverse,light,dark,black,white,custom}
The color scheme to use

**--custom-background-color** <str>, **--custom-bg-color** <str>, **--bg** <str>
Background color for “Custom” color theme

**--custom-foreground-color** <str>, **--custom-fg-color** <str>, **--fg** <str>
Foreground color for “Custom” color theme

**--accent-color** <str>
Accent color to use in the app

**--key-bindings** <loads>
Additional key binding definitions

**--graphics** {none,sixel,kitty,iterm}
The preferred graphics protocol

**--force-graphics, --no-force-graphics**
Force use of specified graphics protocol

**--enable-language-servers, --lsp, --no-enable-language-servers, --no-lsp**
Enable language server support

**--language-servers** <loads>
Language server configurations

**--wrap-cell-outputs, --no-wrap-cell-outputs**
Wrap cell output text.

**--line-numbers, --no-line-numbers**
Show or hide line numbers

**--autoformat, --no-autoformat**
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

**--kernel-name** <str>, **--kernel** <str>
The name of the kernel to start by default

**--record-cell-timing, --no-record-cell-timing**
Should timing data be recorded in cell metadata.

**--max-stored-outputs** <int>
The number of inputs / outputs to store in an in-memory notebook

**--connection-file** <UPath>, **--kernel-connection-file** <UPath>
Attempt to connect to an existing kernel using a JSON connection info file

**--show-file-icons, --no-show-file-icons**
Show file icons in the file manager

**--mouse-support, --no-mouse-support**
Enable or disable mouse support

### 5.9.3 Available Commands

#### Command Available in Euporie Console

**`toggle-version`**

**title**
Toggle version

**description**
Show the version number and exit

**`type-key`**

**title**
Type key

**description**
Enter a key.

**`next-completion`**

**title**
Next completion

**description**
Show the completion menu and select the next completion.

**`previous-completion`**

**title**
Previous completion

**description**
Show the completion menu and select the previous completion.

**`cancel-completion`**

**title**
Cancel completion

**description**
Cancel a completion.

**`accept-completion`**

**title**
Accept completion

**description**
Accept a selected completion.

**`toggle-overwrite-mode`**

**title**
Toggle overwrite mode

**description**
Toggle overwrite when using micro editing mode.

**`start-macro`**

**title**
Start macro

**description**
Start recording a macro.

**`end-macro`**

**title**
End macro

**description**
Stop recording a macro.

**`run-macro`**

**title**
Run macro

**description**
Re-execute the last keyboard macro defined.

**`backspace`**

**title**
Delete previous character

**description**
Delete the character behind the cursor.

**`delete`**

**title**
Delete character

**description**
Delete character before the cursor.

**`backward-kill-word`**

**title**
Delete previous word

**description**
Delete the word behind the cursor, using whitespace as a word boundary.

**`backward-word`**

**title**
Move back one word

**description**
Move back to the start of the current or previous word.

**`forward-word`**

**title**
Move forward one word

**description**
Move forward to the end of the next word.

**`beginning-of-buffer`**

**title**
Move to the beginning of the input

**description**
Move to the start of the buffer.

**`end-of-buffer`**

**title**
Move to the end of the input

**description**
Move to the end of the buffer.

**`scroll-backward`**

**title**
Scroll backward

**description**
Scroll window up.

**`scroll-forward`**

**title**
Scroll forward

**description**
Scroll window down.

**`scroll-half-page-down`**

**title**
Scroll down half a page

**description**
Same as ControlF, but only scroll half a page.

**`scroll-half-page-up`**

**title**
Scroll up half a page

**description**
Same as ControlB, but only scroll half a page.

**`scroll-one-line-down`**

**title**
Scroll down one line

**description**
scroll_offset += 1.

**`scroll-one-line-up`**

**title**
Scroll up one line

**description**
scroll_offset -= 1.

**`move-cursor-left`**

**title**
Move cursor left

**description**
Move back a character, or up a line.

**`move-cursor-right`**

**title**
Move cursor right

**description**
Move forward a character, or down a line.

**`go-to-start-of-line`**

**title**
Go to start of line

**description**
Move the cursor to the start of the line.

**`go-to-end-of-line`**

**title**
Go to end of line

**description**
Move the cursor to the end of the line.

**`go-to-start-of-paragraph`**

**title**
Go to start of paragraph

**description**
Move the cursor to the start of the current paragraph.

**`go-to-end-of-paragraph`**

**title**
Go to end of paragraph

**description**
Move the cursor to the end of the current paragraph.

**`toggle-comment`**

**title**
Toggle comment

**description**
Comment or uncomments the current or selected lines.

**`wrap-selection-""`**

**title**
Wrap selection in “”

**description**
Wraps the current selection with: “”

**wrap-selection-''**

**title**
Wrap selection in ‘’

**description**
Wraps the current selection with: ‘’

**wrap-selection-()**

**title**
Wrap selection in ()

**description**
Wraps the current selection with: ()

**wrap-selection-{}**

**title**
Wrap selection in {}

**description**
Wraps the current selection with: {}

**wrap-selection-[]**

**title**
Wrap selection in []

**description**
Wraps the current selection with: []

**wrap-selection-``**

**title**
Wrap selection in ``

**description**
Wraps the current selection with: ``

**wrap-selection-****

**title**
Wrap selection in **

**description**
Wraps the current selection with: **

**wrap-selection-__**

**title**
Wrap selection in __

**description**
Wraps the current selection with: __

**wrap-selection-<>**

**title**
Wrap selection in <>

**description**
Wraps the current selection with: <>

**`duplicate-line`**

**title**
Duplicate line

**description**
Duplicate the current line.

**`duplicate-selection`**

**title**
Duplicate selection

**description**
Duplicate the current selection.

**`paste-clipboard`**

**title**
Paste

**description**
Pate the clipboard contents, replacing any current selection.

**`copy-selection`**

**title**
Copy

**description**
Add the current selection to the clipboard.

**`cut-selection`**

**title**
Cut

**description**
Remove the current selection and adds it to the clipboard.

**`cut-line`**

**title**
Cut line

**description**
Remove the current line adds it to the clipboard.

**`move-lines-up`**

**title**
Move lines up

**description**
Move the current or selected lines up by one line.

**`move-lines-down`**

**title**
Move lines down

**description**
Move the current or selected lines down by one line.

`accept-line`

**title**
Accept line

**description**
Accept an input.

`newline`

**title**
Newline

**description**
Inert a new line, replacing any selection and indenting if appropriate.

`indent-lines`

**title**
Indent lines

**description**
Inndent the current or selected lines.

`unindent-lines`

**title**
Unindent lines

**description**
Unindent the current or selected lines.

`unindent-line`

**title**
Unindent line

**description**
Unindent the current or selected lines.

`toggle-case`

**title**
Toggle case

**description**
Toggle the case of the current word or selection.

`undo`

**title**
Undo

**description**
Undo the last edit.

`redo`

**title**
Redo

**description**
Redo the last edit.

**`select-all`**

**title**
Select all

**description**
Select all text.

**`start-selection`**

**title**
Start selection

**description**
Start a new selection.

**`extend-selection`**

**title**
Extend selection

**description**
Extend the selection.

**`replace-selection`**

**title**
Replace selection

**description**
Replace selection by what is typed.

**`delete-selection`**

**title**
Delete selection

**description**
Delete the contents of the current selection.

**`cancel-selection`**

**title**
Cancel selection

**description**
Cancel the selection.

**`go-to-matching-bracket`**

**title**
Go to matching bracket

**description**
Go to matching bracket if the cursor is on a paired bracket.

**`accept-suggestion`**

**title**
Accept suggestion

**description**
Accept suggestion.

**`fill-suggestion`**

**title**
Fill suggestion

**description**
Fill partial suggestion.

**`switch-clipboard`**

**title**
Switch clipboard

**description**
Switch the value of the “clipboard” configuration option.

**`set-clipboard-external`**

**title**
Set clipboard to external

**description**
Set the value of the “clipboard” configuration option to “external”

**`set-clipboard-internal`**

**title**
Set clipboard to internal

**description**
Set the value of the “clipboard” configuration option to “internal”

**`set-clipboard-terminal`**

**title**
Set clipboard to terminal

**description**
Set the value of the “clipboard” configuration option to “terminal”

**`switch-log-level`**

**title**
Switch the log level

**description**
Switch the value of the “log_level” configuration option.

**`set-log-level-debug`**

**title**
Set the log level to debug

**description**
Set the value of the “log_level” configuration option to “debug”

**`set-log-level-info`**

**title**
Set the log level to info

**description**
Set the value of the “log_level” configuration option to “info”

**`set-log-level-warning`**

**title**
Set the log level to warning

**description**
Set the value of the “log_level” configuration option to “warning”

**`set-log-level-error`**

**title**
Set the log level to error

**description**
Set the value of the “log_level” configuration option to “error”

**`set-log-level-critical`**

**title**
Set the log level to critical

**description**
Set the value of the “log_level” configuration option to “critical”

**`toggle-show-shadows`**

**title**
Toggle show shadows

**description**
Show or hide shadows under menus and dialogs

**`toggle-show-status-bar`**

**title**
Toggle status bar

**description**
Show the status bar

**`toggle-set-cursor-shape`**

**title**
Toggle set cursor shape

**description**
Whether to set the shape of the cursor depending on the editing mode

**`toggle-cursor-blink`**

**title**
Toggle cursor blink

**description**
Whether to blink the cursor

**`quit`**

**title**
Quit

**description**
Quit euporie.

`close-tab`

**title**
Close tab

**description**
Close the current tab.

`next-tab`

**title**
Next tab

**description**
Switch to the next tab.

`previous-tab`

**title**
Previous tab

**description**
Switch to the previous tab.

`focus-next`

**title**
Focus next

**description**
Focus the next control.

`focus-previous`

**title**
Focus previous

**description**
Focus the previous control.

`clear-screen`

**title**
Clear screen

**description**
Clear the screen and the previous output.

`switch-edit-mode`

**title**
Switch Editor key bindings

**description**
Switch the value of the “edit_mode” configuration option.

`set-edit-mode-micro`

**title**
Set Editor key bindings to micro

**description**
Set the value of the “edit_mode” configuration option to “micro”

**`set-edit-mode-emacs`**

**title**
Set Editor key bindings to emacs

**description**
Set the value of the “edit_mode” configuration option to “emacs”

**`set-edit-mode-vi`**

**title**
Set Editor key bindings to vi

**description**
Set the value of the “edit_mode” configuration option to “vi”

**`switch-tab-size`**

**title**
Switch tab size

**description**
Switch the value of the “tab_size” configuration option.

**`set-syntax-theme-abap`**

**title**
Set syntax theme to abap

**description**
Set the value of the “syntax_theme” configuration option to “abap”

**`set-syntax-theme-algol`**

**title**
Set syntax theme to algol

**description**
Set the value of the “syntax_theme” configuration option to “algol”

**`set-syntax-theme-algol_nu`**

**title**
Set syntax theme to algol_nu

**description**
Set the value of the “syntax_theme” configuration option to “algol_nu”

**`set-syntax-theme-arduino`**

**title**
Set syntax theme to arduino

**description**
Set the value of the “syntax_theme” configuration option to “arduino”

**`set-syntax-theme-autumn`**

**title**
Set syntax theme to autumn

**description**
Set the value of the “syntax_theme” configuration option to “autumn”

**`set-syntax-theme-bw`**

**title**
Set syntax theme to bw

**description**
Set the value of the “syntax_theme” configuration option to “bw”

**`set-syntax-theme-borland`**

**title**
Set syntax theme to borland

**description**
Set the value of the “syntax_theme” configuration option to “borland”

**`set-syntax-theme-coffee`**

**title**
Set syntax theme to coffee

**description**
Set the value of the “syntax_theme” configuration option to “coffee”

**`set-syntax-theme-colorful`**

**title**
Set syntax theme to colorful

**description**
Set the value of the “syntax_theme” configuration option to “colorful”

**`set-syntax-theme-default`**

**title**
Set syntax theme to default

**description**
Set the value of the “syntax_theme” configuration option to “default”

**`set-syntax-theme-dracula`**

**title**
Set syntax theme to dracula

**description**
Set the value of the “syntax_theme” configuration option to “dracula”

**`set-syntax-theme-emacs`**

**title**
Set syntax theme to emacs

**description**
Set the value of the “syntax_theme” configuration option to “emacs”

**`set-syntax-theme-friendly_grayscale`**

**title**
Set syntax theme to friendly_grayscale

**description**
Set the value of the “syntax_theme” configuration option to “friendly_grayscale”

**set-syntax-theme-friendly**

**title**
Set syntax theme to friendly

**description**
Set the value of the "syntax_theme" configuration option to "friendly"

**set-syntax-theme-fruity**

**title**
Set syntax theme to fruity

**description**
Set the value of the "syntax_theme" configuration option to "fruity"

**set-syntax-theme-github-dark**

**title**
Set syntax theme to github-dark

**description**
Set the value of the "syntax_theme" configuration option to "github-dark"

**set-syntax-theme-gruvbox-dark**

**title**
Set syntax theme to gruvbox-dark

**description**
Set the value of the "syntax_theme" configuration option to "gruvbox-dark"

**set-syntax-theme-gruvbox-light**

**title**
Set syntax theme to gruvbox-light

**description**
Set the value of the "syntax_theme" configuration option to "gruvbox-light"

**set-syntax-theme-igor**

**title**
Set syntax theme to igor

**description**
Set the value of the "syntax_theme" configuration option to "igor"

**set-syntax-theme-inkpot**

**title**
Set syntax theme to inkpot

**description**
Set the value of the "syntax_theme" configuration option to "inkpot"

**set-syntax-theme-lightbulb**

**title**
Set syntax theme to lightbulb

**description**
Set the value of the "syntax_theme" configuration option to "lightbulb"

**`set-syntax-theme-lilypond`**

**title**
Set syntax theme to lilypond

**description**
Set the value of the “syntax_theme” configuration option to “lilypond”

**`set-syntax-theme-lovelace`**

**title**
Set syntax theme to lovelace

**description**
Set the value of the “syntax_theme” configuration option to “lovelace”

**`set-syntax-theme-manni`**

**title**
Set syntax theme to manni

**description**
Set the value of the “syntax_theme” configuration option to “manni”

**`set-syntax-theme-material`**

**title**
Set syntax theme to material

**description**
Set the value of the “syntax_theme” configuration option to “material”

**`set-syntax-theme-monokai`**

**title**
Set syntax theme to monokai

**description**
Set the value of the “syntax_theme” configuration option to “monokai”

**`set-syntax-theme-murphy`**

**title**
Set syntax theme to murphy

**description**
Set the value of the “syntax_theme” configuration option to “murphy”

**`set-syntax-theme-native`**

**title**
Set syntax theme to native

**description**
Set the value of the “syntax_theme” configuration option to “native”

**`set-syntax-theme-nord-darker`**

**title**
Set syntax theme to nord-darker

**description**
Set the value of the “syntax_theme” configuration option to “nord-darker”

**`set-syntax-theme-nord`**

**title**
Set syntax theme to nord

**description**
Set the value of the “syntax_theme” configuration option to “nord”

**`set-syntax-theme-one-dark`**

**title**
Set syntax theme to one-dark

**description**
Set the value of the “syntax_theme” configuration option to “one-dark”

**`set-syntax-theme-paraiso-dark`**

**title**
Set syntax theme to paraiso-dark

**description**
Set the value of the “syntax_theme” configuration option to “paraiso-dark”

**`set-syntax-theme-paraiso-light`**

**title**
Set syntax theme to paraiso-light

**description**
Set the value of the “syntax_theme” configuration option to “paraiso-light”

**`set-syntax-theme-pastie`**

**title**
Set syntax theme to pastie

**description**
Set the value of the “syntax_theme” configuration option to “pastie”

**`set-syntax-theme-perldoc`**

**title**
Set syntax theme to perldoc

**description**
Set the value of the “syntax_theme” configuration option to “perldoc”

**`set-syntax-theme-rainbow_dash`**

**title**
Set syntax theme to rainbow_dash

**description**
Set the value of the “syntax_theme” configuration option to “rainbow_dash”

**`set-syntax-theme-rrt`**

**title**
Set syntax theme to rrt

**description**
Set the value of the “syntax_theme” configuration option to “rrt”

**set-syntax-theme-sas**

**title**
Set syntax theme to sas

**description**
Set the value of the "syntax_theme" configuration option to "sas"

**set-syntax-theme-solarized-dark**

**title**
Set syntax theme to solarized-dark

**description**
Set the value of the "syntax_theme" configuration option to "solarized-dark"

**set-syntax-theme-solarized-light**

**title**
Set syntax theme to solarized-light

**description**
Set the value of the "syntax_theme" configuration option to "solarized-light"

**set-syntax-theme-staroffice**

**title**
Set syntax theme to staroffice

**description**
Set the value of the "syntax_theme" configuration option to "staroffice"

**set-syntax-theme-stata-dark**

**title**
Set syntax theme to stata-dark

**description**
Set the value of the "syntax_theme" configuration option to "stata-dark"

**set-syntax-theme-stata-light**

**title**
Set syntax theme to stata-light

**description**
Set the value of the "syntax_theme" configuration option to "stata-light"

**set-syntax-theme-tango**

**title**
Set syntax theme to tango

**description**
Set the value of the "syntax_theme" configuration option to "tango"

**set-syntax-theme-trac**

**title**
Set syntax theme to trac

**description**
Set the value of the "syntax_theme" configuration option to "trac"

**`set-syntax-theme-vim`**

**title**
Set syntax theme to vim

**description**
Set the value of the “syntax_theme” configuration option to “vim”

**`set-syntax-theme-vs`**

**title**
Set syntax theme to vs

**description**
Set the value of the “syntax_theme” configuration option to “vs”

**`set-syntax-theme-xcode`**

**title**
Set syntax theme to xcode

**description**
Set the value of the “syntax_theme” configuration option to “xcode”

**`set-syntax-theme-zenburn`**

**title**
Set syntax theme to zenburn

**description**
Set the value of the “syntax_theme” configuration option to “zenburn”

**`switch-color-depth`**

**title**
Switch color depth

**description**
Switch the value of the “color_depth” configuration option.

**`set-color-depth-1`**

**title**
Set color depth to 1

**description**
Set the value of the “color_depth” configuration option to “1”

**`set-color-depth-4`**

**title**
Set color depth to 4

**description**
Set the value of the “color_depth” configuration option to “4”

**`set-color-depth-8`**

**title**
Set color depth to 8

**description**
Set the value of the “color_depth” configuration option to “8”

**`set-color-depth-24`**

**title**
Set color depth to 24

**description**
Set the value of the “color_depth” configuration option to “24”

**`toggle-multiplexer-passthrough`**

**title**
Toggle multiplexer passthrough

**description**
Use passthrough from within terminal multiplexers

**`switch-color-scheme`**

**title**
Switch color scheme

**description**
Switch the value of the “color_scheme” configuration option.

**`set-color-scheme-default`**

**title**
Set color scheme to default

**description**
Set the value of the “color_scheme” configuration option to “default”

**`set-color-scheme-inverse`**

**title**
Set color scheme to inverse

**description**
Set the value of the “color_scheme” configuration option to “inverse”

**`set-color-scheme-light`**

**title**
Set color scheme to light

**description**
Set the value of the “color_scheme” configuration option to “light”

**`set-color-scheme-dark`**

**title**
Set color scheme to dark

**description**
Set the value of the “color_scheme” configuration option to “dark”

**`set-color-scheme-black`**

**title**
Set color scheme to black

**description**
Set the value of the “color_scheme” configuration option to “black”

**`set-color-scheme-white`**

**title**
Set color scheme to white

**description**
Set the value of the “color_scheme” configuration option to “white”

**`set-color-scheme-custom`**

**title**
Set color scheme to custom

**description**
Set the value of the “color_scheme” configuration option to “custom”

**`switch-graphics`**

**title**
Switch graphics

**description**
Switch the value of the “graphics” configuration option.

**`set-graphics-none`**

**title**
Set graphics to none

**description**
Set the value of the “graphics” configuration option to “none”

**`set-graphics-sixel`**

**title**
Set graphics to sixel

**description**
Set the value of the “graphics” configuration option to “sixel”

**`set-graphics-kitty`**

**title**
Set graphics to kitty

**description**
Set the value of the “graphics” configuration option to “kitty”

**`set-graphics-iterm`**

**title**
Set graphics to iterm

**description**
Set the value of the “graphics” configuration option to “iterm”

**`toggle-force-graphics`**

**title**
Toggle force graphics

**description**
Force use of specified graphics protocol

**`toggle-enable-language-servers`**

**title**
Toggle enable language servers

**description**
Enable language server support

**`scroll-page-down`**

**title**
Scroll page down

**description**
Scroll page down (prefer the cursor at the top of the page, after scrolling).

**`scroll-page-up`**

**title**
Scroll page up

**description**
Scroll page up (prefer the cursor at the bottom of the page, after scrolling).

**`scroll-display-left`**

**title**
Scroll display left

**description**
Scroll the display up one line.

**`scroll-display-right`**

**title**
Scroll display right

**description**
Scroll the display down one line.

**`scroll-display-up`**

**title**
Scroll display up

**description**
Scroll the display up one line.

**`scroll-display-down`**

**title**
Scroll display down

**description**
Scroll the display down one line.

**`page-up-display`**

**title**
Page up display

**description**
Scroll the display up one page.

**`page-down-display`**

**title**
Page down display

**description**
Scroll the display down one page.

**`go-to-start-of-display`**

**title**
Go to start of display

**description**
Scroll the display to the top.

**`go-to-end-of-display`**

**title**
Go to end of display

**description**
Scroll the display down one page.

**`toggle-wrap-cell-outputs`**

**title**
Toggle wrap cell outputs

**description**
Wrap cell output text.

**`close-pager`**

**title**
Close pager

**description**
Close the pager.

**`toggle-line-numbers`**

**title**
Toggle line numbers

**description**
Show or hide line numbers

**`toggle-autoformat`**

**title**
Toggle autoformat

**description**
Automatically re-format code cells when run

**`toggle-autocomplete`**

**title**
Toggle autocomplete

**description**
Provide completions suggestions automatically

**`toggle-autosuggest`**

**title**
Toggle autosuggest

**description**
Provide line completion suggestions

**`toggle-autoinspect`**

**title**
Toggle autoinspect

**description**
Display contextual help automatically

**`show-contextual-help`**

**title**
Show contextual help

**description**
Display contextual help.

**`history-prev`**

**title**
History prev

**description**
Get the previous history entry.

**`history-next`**

**title**
History next

**description**
Get the next history entry.

**`reformat-input`**

**title**
Reformat input

**description**
Format the contents of the current input field.

**`refresh-tab`**

**title**
Refresh the current tab

**description**
Reload the tab contents and reset the tab.

**`reset-tab`**

**title**
Reset the current tab

**description**
Reset the current tab

**`save-file`**

**title**
Save file

**description**
Save the current file.

**`change-kernel`**

**title**
Change kernel

**description**
Change the notebook's kernel.

**`toggle-record-cell-timing`**

**title**
Toggle cell timing recording

**description**
Should timing data be recorded in cell metadata.

**`accept-input`**

**title**
Accept input

**description**
Accept the current console input.

**`clear-input`**

**title**
Clear input

**description**
Clear the console input.

**`run-input`**

**title**
Run input

**description**
Run the console input.

**`interrupt-kernel`**

**title**
Interrupt kernel

**description**
Interrupt the notebook's kernel.

**`cc-interrupt-kernel`**

**title**
Cc interrupt kernel

**description**
Interrupt the notebook's kernel.

**`restart-kernel`**

**title**
Restart kernel

**description**
Restart the notebook's kernel.

**`end-of-file`**

**title**
End of file

**description**
Signals the end of the input, causing the console to exit.

**`switch-max-stored-outputs`**

**title**
Switch max stored outputs

**description**
Switch the value of the "max_stored_outputs" configuration option.

**`toggle-show-file-icons`**

**title**
Toggle File icons

**description**
Show file icons in the file manager

**`about`**

**title**
About

**description**
Show the about dialog.

**`open-file`**

**title**
Open file

**description**
Open a file.

**`save-as`**

**title**
Save as

**description**
Save the current file at a new location.

**`keyboard-shortcuts`**

**title**
Keyboard shortcuts

**description**
Display details of registered key-bindings in a dialog.

**`toggle-command-palette`**

**title**
Toggle command palette

**description**
Show the command palette.

**`show-command-palette`**

**title**
Show command palette

**description**
Show the command palette.

**`hide-command-palette`**

**title**
Hide command palette

**description**
Hide the command palette.

**`find`**

**title**
Find

**description**
Enter search mode.

**`find-next`**

**title**
Find next

**description**
Find the next search match.

**`find-previous`**

**title**
Find previous

**description**
Find the previous search match.

**`stop-search`**

**title**
Stop search

**description**
Abort the search.

**`accept-search`**

**title**
Accept search

**description**
Accept the search input.

**convert-to-notebook**

**title**
Convert to notebook

**description**
Convert the current console session to a notebook.

**toggle-mouse-support**

**title**
Toggle mouse support

**description**
Enable or disable mouse support

## 5.10 Notebook Preview

Euporie can be used to render notebooks in the terminal, which is useful for quickly previewing notebook files.

For more information about the command line options available for the `preview` sub-command, see the command line reference for the preview-subcommand.

### 5.10.1 User Guide

#### Preview a Notebook in the Terminal

To print a notebook to the terminal, run:

```
$ euporie-preview notebook.ipynb
```

#### Preview a Notebook in the System Pager

To view a notebook in the system pager, run:

```
$ euporie-preview --page notebook.ipynb
```

You can also pipe the output to the pager of your choice:

```
$ euporie-preview --color-depth=24 notebook.ipynb | bat
```

---

**Note:** By default euporie will select a color-depth to use which is suitable for the environment it is running in. This means if you are piping its output, the color depth will be set to `1`. If your pager supports colored output, you can manually specify the color-depth with the `color_depth` configuration option.

---

### Run a Notebook Before Previewing

To run a notebook before the preview is generated, use the `--run` flag:

```
$ euporie-preview --run notebook.ipynb
```

### Preview a subset of cells

To show a subset of the cells in the notebook, the `--cell-start` and `--cell-end` flags can be used:

```
$ euporie-preview --cell-start=3 --cell-end=6 notebook.ipynb
```

### Save a Notebook After Running

To save a notebook after it has been run, use the `--save` flag with the `--run` flag:

```
$ euporie-preview --run --save notebook.ipynb
```

### Use as a previewer with `ranger`

Euporie can be used to preview notebook files in terminal file managers like **`ranger`**.

To configure **`ranger`** for this, add the following to the `handle_extension` function in your `scope.sh` file:

```
# ...

handle_extension() {
    case "${FILE_EXTENSION_LOWER}" in

        # ...

        ## Notebook
        ipynb)
            euporie-preview --color-depth=8 "${FILE_PATH}" && exit 4
    esac
}

# ...
```

You can also add the following line to your `rifle.conf` file if you want notebook files to open in euporie:

```
ext ipynb,          has euporie-notebook,      terminal = euporie-notebook "$@"
```

### 5.10.2 Command Line Interface

#### Usage

```
$ euporie-hub [-h] [--version] [--clipboard {external,internal,terminal}]
              [--log-file [str]]
              [--log-level {debug,info,warning,error,critical}]
              [--log-config str] [--show-shadows | --no-show-shadows]
              [--show-status-bar | --no-show-status-bar]
              [--set-cursor-shape | --no-set-cursor-shape]
              [--cursor-blink | --no-cursor-blink]
              [--edit-mode {micro,emacs,vi}] [--tab-size int]
              [--terminal-polling-interval float] [--formatters loads]
              [--syntax-theme str] [--color-depth {1,4,8,24}]
              [--multiplexer-passthrough | --no-multiplexer-passthrough]
              [--color-scheme {default,inverse,light,dark,black,white,custom}]
              [--custom-background-color str]
              [--custom-foreground-color str] [--accent-color str]
              [--key-bindings loads]
              [--graphics {none,sixel,kitty,iterm}]
              [--force-graphics | --no-force-graphics]
              [--enable-language-servers | --lsp | --no-enable-language-servers␣
↪| --no-lsp]
              [--language-servers loads] [--app {notebook,console}]
              [--host str] [--port int] [--host-keys [UPath ...]]
              [--client-keys [UPath ...]] [--auth | --no-auth]
              [UPath ...]
```

#### Positional Arguments

**<UPath> ...**
: List of file names to open

#### Optional Arguments

**-h, --help**
: show this help message and exit

**--version, -V**
: Show the version number and exit

**--clipboard** {external,internal,terminal}
: The preferred clipboard access method

**--log-file** <str>
: File path for logs

**--log-level** {debug,info,warning,error,critical}
: Set the log level

**--log-config** <str>
: Additional logging configuration

**--show-shadows, --no-show-shadows**
Show or hide shadows under menus and dialogs

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

**--set-cursor-shape, --no-set-cursor-shape**
Whether to set the shape of the cursor depending on the editing mode

**--cursor-blink, --no-cursor-blink**
Whether to blink the cursor

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

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

**--terminal-polling-interval** <float>
Time between terminal colour queries

**--formatters** <loads>
List of external code formatters

**--syntax-theme** <str>
Syntax highlighting theme

**--color-depth** {1,4,8,24}
The color depth to use

**--multiplexer-passthrough, --no-multiplexer-passthrough**
Use passthrough from within terminal multiplexers

**--color-scheme** {default,inverse,light,dark,black,white,custom}
The color scheme to use

**--custom-background-color** <str>, **--custom-bg-color** <str>, **--bg** <str>
Background color for "Custom" color theme

**--custom-foreground-color** <str>, **--custom-fg-color** <str>, **--fg** <str>
Foreground color for "Custom" color theme

**--accent-color** <str>
Accent color to use in the app

**--key-bindings** <loads>
Additional key binding definitions

**--graphics** {none,sixel,kitty,iterm}
The preferred graphics protocol

**--force-graphics, --no-force-graphics**
Force use of specified graphics protocol

**--enable-language-servers, --lsp, --no-enable-language-servers, --no-lsp**
Enable language server support

**--language-servers** <loads>
Language server configurations

**--app** {notebook,console}
App to run under euporie hub

**--host** <str>
The host address to bind to

**--port** <int>
The port for the ssh server to use

**--host-keys** <UPath> ...
Host keys to use for the SSH server

**--client-keys** <UPath> ...
Client public keys authorized to connect

**--auth, --no-auth**
Allow unauthenticated access to euporie hub

---

### 5.10.3 Available Commands

#### Command Available in Euporie Preview

**toggle-version**

**title**
Toggle version

**description**
Show the version number and exit

**switch-clipboard**

**title**
Switch clipboard

**description**
Switch the value of the "clipboard" configuration option.

**set-clipboard-external**

**title**
Set clipboard to external

**description**
Set the value of the "clipboard" configuration option to "external"

**set-clipboard-internal**

**title**
Set clipboard to internal

**description**
Set the value of the "clipboard" configuration option to "internal"

**`set-clipboard-terminal`**

**title**
Set clipboard to terminal

**description**
Set the value of the "clipboard" configuration option to "terminal"

**`switch-log-level`**

**title**
Switch the log level

**description**
Switch the value of the "log_level" configuration option.

**`set-log-level-debug`**

**title**
Set the log level to debug

**description**
Set the value of the "log_level" configuration option to "debug"

**`set-log-level-info`**

**title**
Set the log level to info

**description**
Set the value of the "log_level" configuration option to "info"

**`set-log-level-warning`**

**title**
Set the log level to warning

**description**
Set the value of the "log_level" configuration option to "warning"

**`set-log-level-error`**

**title**
Set the log level to error

**description**
Set the value of the "log_level" configuration option to "error"

**`set-log-level-critical`**

**title**
Set the log level to critical

**description**
Set the value of the "log_level" configuration option to "critical"

**`toggle-show-shadows`**

**title**
Toggle show shadows

**description**
Show or hide shadows under menus and dialogs

`toggle-show-status-bar`

**title**
Toggle status bar

**description**
Show the status bar

`toggle-set-cursor-shape`

**title**
Toggle set cursor shape

**description**
Whether to set the shape of the cursor depending on the editing mode

`toggle-cursor-blink`

**title**
Toggle cursor blink

**description**
Whether to blink the cursor

`quit`

**title**
Quit

**description**
Quit euporie.

`close-tab`

**title**
Close tab

**description**
Close the current tab.

`next-tab`

**title**
Next tab

**description**
Switch to the next tab.

`previous-tab`

**title**
Previous tab

**description**
Switch to the previous tab.

`focus-next`

**title**
Focus next

**description**
Focus the next control.

**`focus-previous`**

**title**
Focus previous

**description**
Focus the previous control.

**`clear-screen`**

**title**
Clear screen

**description**
Clear the screen.

**`switch-edit-mode`**

**title**
Switch Editor key bindings

**description**
Switch the value of the “edit_mode” configuration option.

**`set-edit-mode-micro`**

**title**
Set Editor key bindings to micro

**description**
Set the value of the “edit_mode” configuration option to “micro”

**`set-edit-mode-emacs`**

**title**
Set Editor key bindings to emacs

**description**
Set the value of the “edit_mode” configuration option to “emacs”

**`set-edit-mode-vi`**

**title**
Set Editor key bindings to vi

**description**
Set the value of the “edit_mode” configuration option to “vi”

**`switch-tab-size`**

**title**
Switch tab size

**description**
Switch the value of the “tab_size” configuration option.

**`set-syntax-theme-abap`**

**title**
Set syntax theme to abap

**description**
Set the value of the “syntax_theme” configuration option to “abap”

**set-syntax-theme-algol**

**title**
Set syntax theme to algol

**description**
Set the value of the “syntax_theme” configuration option to “algol”

**set-syntax-theme-algol_nu**

**title**
Set syntax theme to algol_nu

**description**
Set the value of the “syntax_theme” configuration option to “algol_nu”

**set-syntax-theme-arduino**

**title**
Set syntax theme to arduino

**description**
Set the value of the “syntax_theme” configuration option to “arduino”

**set-syntax-theme-autumn**

**title**
Set syntax theme to autumn

**description**
Set the value of the “syntax_theme” configuration option to “autumn”

**set-syntax-theme-bw**

**title**
Set syntax theme to bw

**description**
Set the value of the “syntax_theme” configuration option to “bw”

**set-syntax-theme-borland**

**title**
Set syntax theme to borland

**description**
Set the value of the “syntax_theme” configuration option to “borland”

**set-syntax-theme-coffee**

**title**
Set syntax theme to coffee

**description**
Set the value of the “syntax_theme” configuration option to “coffee”

**set-syntax-theme-colorful**

**title**
Set syntax theme to colorful

**description**
Set the value of the “syntax_theme” configuration option to “colorful”

**`set-syntax-theme-default`**

**title**
Set syntax theme to default

**description**
Set the value of the "syntax_theme" configuration option to "default"

**`set-syntax-theme-dracula`**

**title**
Set syntax theme to dracula

**description**
Set the value of the "syntax_theme" configuration option to "dracula"

**`set-syntax-theme-emacs`**

**title**
Set syntax theme to emacs

**description**
Set the value of the "syntax_theme" configuration option to "emacs"

**`set-syntax-theme-friendly_grayscale`**

**title**
Set syntax theme to friendly_grayscale

**description**
Set the value of the "syntax_theme" configuration option to "friendly_grayscale"

**`set-syntax-theme-friendly`**

**title**
Set syntax theme to friendly

**description**
Set the value of the "syntax_theme" configuration option to "friendly"

**`set-syntax-theme-fruity`**

**title**
Set syntax theme to fruity

**description**
Set the value of the "syntax_theme" configuration option to "fruity"

**`set-syntax-theme-github-dark`**

**title**
Set syntax theme to github-dark

**description**
Set the value of the "syntax_theme" configuration option to "github-dark"

**`set-syntax-theme-gruvbox-dark`**

**title**
Set syntax theme to gruvbox-dark

**description**
Set the value of the "syntax_theme" configuration option to "gruvbox-dark"

**`set-syntax-theme-gruvbox-light`**

**title**
Set syntax theme to gruvbox-light

**description**
Set the value of the “syntax_theme” configuration option to “gruvbox-light”

**`set-syntax-theme-igor`**

**title**
Set syntax theme to igor

**description**
Set the value of the “syntax_theme” configuration option to “igor”

**`set-syntax-theme-inkpot`**

**title**
Set syntax theme to inkpot

**description**
Set the value of the “syntax_theme” configuration option to “inkpot”

**`set-syntax-theme-lightbulb`**

**title**
Set syntax theme to lightbulb

**description**
Set the value of the “syntax_theme” configuration option to “lightbulb”

**`set-syntax-theme-lilypond`**

**title**
Set syntax theme to lilypond

**description**
Set the value of the “syntax_theme” configuration option to “lilypond”

**`set-syntax-theme-lovelace`**

**title**
Set syntax theme to lovelace

**description**
Set the value of the “syntax_theme” configuration option to “lovelace”

**`set-syntax-theme-manni`**

**title**
Set syntax theme to manni

**description**
Set the value of the “syntax_theme” configuration option to “manni”

**`set-syntax-theme-material`**

**title**
Set syntax theme to material

**description**
Set the value of the “syntax_theme” configuration option to “material”

**`set-syntax-theme-monokai`**

**title**
Set syntax theme to monokai

**description**
Set the value of the “syntax_theme” configuration option to “monokai”

**`set-syntax-theme-murphy`**

**title**
Set syntax theme to murphy

**description**
Set the value of the “syntax_theme” configuration option to “murphy”

**`set-syntax-theme-native`**

**title**
Set syntax theme to native

**description**
Set the value of the “syntax_theme” configuration option to “native”

**`set-syntax-theme-nord-darker`**

**title**
Set syntax theme to nord-darker

**description**
Set the value of the “syntax_theme” configuration option to “nord-darker”

**`set-syntax-theme-nord`**

**title**
Set syntax theme to nord

**description**
Set the value of the “syntax_theme” configuration option to “nord”

**`set-syntax-theme-one-dark`**

**title**
Set syntax theme to one-dark

**description**
Set the value of the “syntax_theme” configuration option to “one-dark”

**`set-syntax-theme-paraiso-dark`**

**title**
Set syntax theme to paraiso-dark

**description**
Set the value of the “syntax_theme” configuration option to “paraiso-dark”

**`set-syntax-theme-paraiso-light`**

**title**
Set syntax theme to paraiso-light

**description**
Set the value of the “syntax_theme” configuration option to “paraiso-light”

**set-syntax-theme-pastie**

**title**
Set syntax theme to pastie

**description**
Set the value of the “syntax_theme” configuration option to “pastie”

**set-syntax-theme-perldoc**

**title**
Set syntax theme to perldoc

**description**
Set the value of the “syntax_theme” configuration option to “perldoc”

**set-syntax-theme-rainbow_dash**

**title**
Set syntax theme to rainbow_dash

**description**
Set the value of the “syntax_theme” configuration option to “rainbow_dash”

**set-syntax-theme-rrt**

**title**
Set syntax theme to rrt

**description**
Set the value of the “syntax_theme” configuration option to “rrt”

**set-syntax-theme-sas**

**title**
Set syntax theme to sas

**description**
Set the value of the “syntax_theme” configuration option to “sas”

**set-syntax-theme-solarized-dark**

**title**
Set syntax theme to solarized-dark

**description**
Set the value of the “syntax_theme” configuration option to “solarized-dark”

**set-syntax-theme-solarized-light**

**title**
Set syntax theme to solarized-light

**description**
Set the value of the “syntax_theme” configuration option to “solarized-light”

**set-syntax-theme-staroffice**

**title**
Set syntax theme to staroffice

**description**
Set the value of the “syntax_theme” configuration option to “staroffice”

**`set-syntax-theme-stata-dark`**

**title**
Set syntax theme to stata-dark

**description**
Set the value of the "syntax_theme" configuration option to "stata-dark"

**`set-syntax-theme-stata-light`**

**title**
Set syntax theme to stata-light

**description**
Set the value of the "syntax_theme" configuration option to "stata-light"

**`set-syntax-theme-tango`**

**title**
Set syntax theme to tango

**description**
Set the value of the "syntax_theme" configuration option to "tango"

**`set-syntax-theme-trac`**

**title**
Set syntax theme to trac

**description**
Set the value of the "syntax_theme" configuration option to "trac"

**`set-syntax-theme-vim`**

**title**
Set syntax theme to vim

**description**
Set the value of the "syntax_theme" configuration option to "vim"

**`set-syntax-theme-vs`**

**title**
Set syntax theme to vs

**description**
Set the value of the "syntax_theme" configuration option to "vs"

**`set-syntax-theme-xcode`**

**title**
Set syntax theme to xcode

**description**
Set the value of the "syntax_theme" configuration option to "xcode"

**`set-syntax-theme-zenburn`**

**title**
Set syntax theme to zenburn

**description**
Set the value of the "syntax_theme" configuration option to "zenburn"

**`switch-color-depth`**

**title**
Switch color depth

**description**
Switch the value of the “color_depth” configuration option.

**`set-color-depth-1`**

**title**
Set color depth to 1

**description**
Set the value of the “color_depth” configuration option to “1”

**`set-color-depth-4`**

**title**
Set color depth to 4

**description**
Set the value of the “color_depth” configuration option to “4”

**`set-color-depth-8`**

**title**
Set color depth to 8

**description**
Set the value of the “color_depth” configuration option to “8”

**`set-color-depth-24`**

**title**
Set color depth to 24

**description**
Set the value of the “color_depth” configuration option to “24”

**`toggle-multiplexer-passthrough`**

**title**
Toggle multiplexer passthrough

**description**
Use passthrough from within terminal multiplexers

**`switch-color-scheme`**

**title**
Switch color scheme

**description**
Switch the value of the “color_scheme” configuration option.

**`set-color-scheme-default`**

**title**
Set color scheme to default

**description**
Set the value of the “color_scheme” configuration option to “default”

**`set-color-scheme-inverse`**

**title**
Set color scheme to inverse

**description**
Set the value of the “color_scheme” configuration option to “inverse”

**`set-color-scheme-light`**

**title**
Set color scheme to light

**description**
Set the value of the “color_scheme” configuration option to “light”

**`set-color-scheme-dark`**

**title**
Set color scheme to dark

**description**
Set the value of the “color_scheme” configuration option to “dark”

**`set-color-scheme-black`**

**title**
Set color scheme to black

**description**
Set the value of the “color_scheme” configuration option to “black”

**`set-color-scheme-white`**

**title**
Set color scheme to white

**description**
Set the value of the “color_scheme” configuration option to “white”

**`set-color-scheme-custom`**

**title**
Set color scheme to custom

**description**
Set the value of the “color_scheme” configuration option to “custom”

**`switch-graphics`**

**title**
Switch graphics

**description**
Switch the value of the “graphics” configuration option.

**`set-graphics-none`**

**title**
Set graphics to none

**description**
Set the value of the “graphics” configuration option to “none”

**`set-graphics-sixel`**

**title**
Set graphics to sixel

**description**
Set the value of the “graphics” configuration option to “sixel”

**`set-graphics-kitty`**

**title**
Set graphics to kitty

**description**
Set the value of the “graphics” configuration option to “kitty”

**`set-graphics-iterm`**

**title**
Set graphics to iterm

**description**
Set the value of the “graphics” configuration option to “iterm”

**`toggle-force-graphics`**

**title**
Toggle force graphics

**description**
Force use of specified graphics protocol

**`toggle-enable-language-servers`**

**title**
Toggle enable language servers

**description**
Enable language server support

**`type-key`**

**title**
Type key

**description**
Enter a key.

**`next-completion`**

**title**
Next completion

**description**
Show the completion menu and select the next completion.

**`previous-completion`**

**title**
Previous completion

**description**
Show the completion menu and select the previous completion.

**`cancel-completion`**

**title**
Cancel completion

**description**
Cancel a completion.

**`accept-completion`**

**title**
Accept completion

**description**
Accept a selected completion.

**`toggle-overwrite-mode`**

**title**
Toggle overwrite mode

**description**
Toggle overwrite when using micro editing mode.

**`start-macro`**

**title**
Start macro

**description**
Start recording a macro.

**`end-macro`**

**title**
End macro

**description**
Stop recording a macro.

**`run-macro`**

**title**
Run macro

**description**
Re-execute the last keyboard macro defined.

**`backspace`**

**title**
Delete previous character

**description**
Delete the character behind the cursor.

**`delete`**

**title**
Delete character

**description**
Delete character before the cursor.

**`backward-kill-word`**

**title**
Delete previous word

**description**
Delete the word behind the cursor, using whitespace as a word boundary.

**`backward-word`**

**title**
Move back one word

**description**
Move back to the start of the current or previous word.

**`forward-word`**

**title**
Move forward one word

**description**
Move forward to the end of the next word.

**`beginning-of-buffer`**

**title**
Move to the beginning of the input

**description**
Move to the start of the buffer.

**`end-of-buffer`**

**title**
Move to the end of the input

**description**
Move to the end of the buffer.

**`scroll-backward`**

**title**
Scroll backward

**description**
Scroll window up.

**`scroll-forward`**

**title**
Scroll forward

**description**
Scroll window down.

**`scroll-half-page-down`**

**title**
Scroll down half a page

**description**
Same as ControlF, but only scroll half a page.

**`scroll-half-page-up`**

**title**
Scroll up half a page

**description**
Same as ControlB, but only scroll half a page.

**`scroll-one-line-down`**

**title**
Scroll down one line

**description**
scroll_offset += 1.

**`scroll-one-line-up`**

**title**
Scroll up one line

**description**
scroll_offset -= 1.

**`move-cursor-left`**

**title**
Move cursor left

**description**
Move back a character, or up a line.

**`move-cursor-right`**

**title**
Move cursor right

**description**
Move forward a character, or down a line.

**`go-to-start-of-line`**

**title**
Go to start of line

**description**
Move the cursor to the start of the line.

**`go-to-end-of-line`**

**title**
Go to end of line

**description**
Move the cursor to the end of the line.

**`go-to-start-of-paragraph`**

**title**
Go to start of paragraph

**description**
Move the cursor to the start of the current paragraph.

**go-to-end-of-paragraph**

**title**
Go to end of paragraph

**description**
Move the cursor to the end of the current paragraph.

**toggle-comment**

**title**
Toggle comment

**description**
Comment or uncomments the current or selected lines.

**wrap-selection-""**

**title**
Wrap selection in “”

**description**
Wraps the current selection with: “”

**wrap-selection-''**

**title**
Wrap selection in ‘

**description**
Wraps the current selection with: ‘

**wrap-selection-()**

**title**
Wrap selection in ()

**description**
Wraps the current selection with: ()

**wrap-selection-{}**

**title**
Wrap selection in {}

**description**
Wraps the current selection with: {}

**wrap-selection-[]**

**title**
Wrap selection in []

**description**
Wraps the current selection with: []

**wrap-selection-``**

**title**
Wrap selection in ``

**description**
Wraps the current selection with: ``

**wrap-selection-****

**title**
Wrap selection in **

**description**
Wraps the current selection with: **

**wrap-selection-__**

**title**
Wrap selection in __

**description**
Wraps the current selection with: __

**wrap-selection-<>**

**title**
Wrap selection in <>

**description**
Wraps the current selection with: <>

**duplicate-line**

**title**
Duplicate line

**description**
Duplicate the current line.

**duplicate-selection**

**title**
Duplicate selection

**description**
Duplicate the current selection.

**paste-clipboard**

**title**
Paste

**description**
Pate the clipboard contents, replacing any current selection.

**copy-selection**

**title**
Copy

**description**
Add the current selection to the clipboard.

**cut-selection**

**title**
Cut

**description**
Remove the current selection and adds it to the clipboard.

`cut-line`

**title**
Cut line

**description**
Remove the current line adds it to the clipboard.

`move-lines-up`

**title**
Move lines up

**description**
Move the current or selected lines up by one line.

`move-lines-down`

**title**
Move lines down

**description**
Move the current or selected lines down by one line.

`accept-line`

**title**
Accept line

**description**
Accept an input.

`newline`

**title**
Newline

**description**
Inert a new line, replacing any selection and indenting if appropriate.

`indent-lines`

**title**
Indent lines

**description**
Inndent the current or selected lines.

`unindent-lines`

**title**
Unindent lines

**description**
Unindent the current or selected lines.

`unindent-line`

**title**
Unindent line

**description**
Unindent the current or selected lines.

`toggle-case`

**title**
Toggle case

**description**
Toggle the case of the current word or selection.

`undo`

**title**
Undo

**description**
Undo the last edit.

`redo`

**title**
Redo

**description**
Redo the last edit.

`select-all`

**title**
Select all

**description**
Select all text.

`start-selection`

**title**
Start selection

**description**
Start a new selection.

`extend-selection`

**title**
Extend selection

**description**
Extend the selection.

`replace-selection`

**title**
Replace selection

**description**
Replace selection by what is typed.

`delete-selection`

**title**
Delete selection

**description**
Delete the contents of the current selection.

`cancel-selection`

**title**
Cancel selection

**description**
Cancel the selection.

`go-to-matching-bracket`

**title**
Go to matching bracket

**description**
Go to matching bracket if the cursor is on a paired bracket.

`accept-suggestion`

**title**
Accept suggestion

**description**
Accept suggestion.

`fill-suggestion`

**title**
Fill suggestion

**description**
Fill partial suggestion.

`scroll-page-down`

**title**
Scroll page down

**description**
Scroll page down (prefer the cursor at the top of the page, after scrolling).

`scroll-page-up`

**title**
Scroll page up

**description**
Scroll page up (prefer the cursor at the bottom of the page, after scrolling).

`scroll-display-left`

**title**
Scroll display left

**description**
Scroll the display up one line.

`scroll-display-right`

**title**
Scroll display right

**description**
Scroll the display down one line.

**`scroll-display-up`**

**title**
Scroll display up

**description**
Scroll the display up one line.

**`scroll-display-down`**

**title**
Scroll display down

**description**
Scroll the display down one line.

**`page-up-display`**

**title**
Page up display

**description**
Scroll the display up one page.

**`page-down-display`**

**title**
Page down display

**description**
Scroll the display down one page.

**`go-to-start-of-display`**

**title**
Go to start of display

**description**
Scroll the display to the top.

**`go-to-end-of-display`**

**title**
Go to end of display

**description**
Scroll the display down one page.

**`toggle-wrap-cell-outputs`**

**title**
Toggle wrap cell outputs

**description**
Wrap cell output text.

**`close-pager`**

**title**
Close pager

**description**
Close the pager.

**`toggle-line-numbers`**

**title**
Toggle line numbers

**description**
Show or hide line numbers

**`toggle-autoformat`**

**title**
Toggle autoformat

**description**
Automatically re-format code cells when run

**`toggle-autocomplete`**

**title**
Toggle autocomplete

**description**
Provide completions suggestions automatically

**`toggle-autosuggest`**

**title**
Toggle autosuggest

**description**
Provide line completion suggestions

**`toggle-autoinspect`**

**title**
Toggle autoinspect

**description**
Display contextual help automatically

**`show-contextual-help`**

**title**
Show contextual help

**description**
Display contextual help.

**`history-prev`**

**title**
History prev

**description**
Get the previous history entry.

**`history-next`**

**title**
History next

**description**
Get the next history entry.

**`reformat-input`**

**title**
Reformat input

**description**
Format the contents of the current input field.

**`refresh-tab`**

**title**
Refresh the current tab

**description**
Reload the tab contents and reset the tab.

**`reset-tab`**

**title**
Reset the current tab

**description**
Reset the current tab

**`save-file`**

**title**
Save file

**description**
Save the current file.

**`change-kernel`**

**title**
Change kernel

**description**
Change the notebook's kernel.

**`toggle-record-cell-timing`**

**title**
Toggle cell timing recording

**description**
Should timing data be recorded in cell metadata.

**`toggle-show-cell-borders`**

**title**
Toggle cell borders

**description**
Show or hide cell borders.

**`toggle-save-widget-state`**

**title**
Toggle save widget state

**description**
Save a notebook's widget state in the notebook metadata

**switch-max-notebook-width**

**title**
Switch max notebook width

**description**
Switch the value of the “max_notebook_width” configuration option.

**toggle-expand**

**title**
Toggle expand

**description**
Use the full width to display notebooks

**toggle-run**

**title**
Toggle run

**description**
Run the notebook files when loaded

**toggle-save**

**title**
Toggle save

**description**
Save the notebook after running it

**toggle-show-filenames**

**title**
Toggle show filenames

**description**
Show the notebook filenames when previewing multiple notebooks

**switch-cell-start**

**title**
Switch cell start

**description**
Switch the value of the “cell_start” configuration option.

**switch-cell-stop**

**title**
Switch cell stop

**description**
Switch the value of the “cell_stop” configuration option.

**toggle-page**

**title**
Toggle page

**description**
Pass output to pager

## 5.11 Euporie Hub

Euporie Hub allows a group of users to share the power of euporie over a multi-user SSH server, akin to JupyterHub.

It is launched by running **`euporie`** with the `hub` subcommand:

```
$ euporie hub
```

### 5.11.1 User Guide

Euporie hub requires a set of SSH host keys to run, which can be generated as follows:

```
$ ssh-keygen -t ed25519 -f ssh_host_ed25519_key
```

Euporie hub can then be run as follows, specifying the host and client key locations:

```
$ euporie-hub --host-keys ssh_host_ed25519_key --client-keys ~/.ssh/authorized_keys
```

You should now be able to log on using one of the SSH keys which appear in your `authorized_keys` file.

---

### 5.11.2 Command Line Interface

#### Usage

```
$ euporie-hub [-h] [--version] [--clipboard {external,internal,terminal}]
                  [--log-file [str]]
                  [--log-level {debug,info,warning,error,critical}]
                  [--log-config str] [--show-shadows | --no-show-shadows]
                  [--show-status-bar | --no-show-status-bar]
                  [--set-cursor-shape | --no-set-cursor-shape]
                  [--cursor-blink | --no-cursor-blink]
                  [--edit-mode {micro,emacs,vi}] [--tab-size int]
                  [--terminal-polling-interval float] [--formatters loads]
                  [--syntax-theme str] [--color-depth {1,4,8,24}]
                  [--multiplexer-passthrough | --no-multiplexer-passthrough]
                  [--color-scheme {default,inverse,light,dark,black,white,custom}]
                  [--custom-background-color str]
                  [--custom-foreground-color str] [--accent-color str]
                  [--key-bindings loads]
                  [--graphics {none,sixel,kitty,iterm}]
                  [--force-graphics | --no-force-graphics]
                  [--enable-language-servers | --lsp | --no-enable-language-servers␣
↪| --no-lsp]
                  [--language-servers loads] [--app {notebook,console}]
                  [--host str] [--port int] [--host-keys [UPath ...]]
                  [--client-keys [UPath ...]] [--auth | --no-auth]
                  [UPath ...]
```

### Positional Arguments

**<UPath> ...**
List of file names to open

### Optional Arguments

**-h, --help**
show this help message and exit

**--version, -V**
Show the version number and exit

**--clipboard** {external,internal,terminal}
The preferred clipboard access method

**--log-file** <str>
File path for logs

**--log-level** {debug,info,warning,error,critical}
Set the log level

**--log-config** <str>
Additional logging configuration

**--show-shadows, --no-show-shadows**
Show or hide shadows under menus and dialogs

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

**--set-cursor-shape, --no-set-cursor-shape**
Whether to set the shape of the cursor depending on the editing mode

**--cursor-blink, --no-cursor-blink**
Whether to blink the cursor

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

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

**--terminal-polling-interval** <float>
Time between terminal colour queries

**--formatters** <loads>
List of external code formatters

**--syntax-theme** <str>
Syntax highlighting theme

**--color-depth** {1,4,8,24}
The color depth to use

**--multiplexer-passthrough, --no-multiplexer-passthrough**
Use passthrough from within terminal multiplexers

**--color-scheme** {default,inverse,light,dark,black,white,custom}
The color scheme to use

**--custom-background-color** <str>, **--custom-bg-color** <str>, **--bg** <str>
Background color for "Custom" color theme

**--custom-foreground-color** <str>, **--custom-fg-color** <str>, **--fg** <str>
Foreground color for "Custom" color theme

**--accent-color** <str>
Accent color to use in the app

**--key-bindings** <loads>
Additional key binding definitions

**--graphics** {none,sixel,kitty,iterm}
The preferred graphics protocol

**--force-graphics**, **--no-force-graphics**
Force use of specified graphics protocol

**--enable-language-servers**, **--lsp**, **--no-enable-language-servers**, **--no-lsp**
Enable language server support

**--language-servers** <loads>
Language server configurations

**--app** {notebook,console}
App to run under euporie hub

**--host** <str>
The host address to bind to

**--port** <int>
The port for the ssh server to use

**--host-keys** <UPath> ...
Host keys to use for the SSH server

**--client-keys** <UPath> ...
Client public keys authorized to connect

**--auth**, **--no-auth**
Allow unauthenticated access to euporie hub

### 5.11.3 Available Commands

#### Command Available in Euporie Hub

**toggle-version**

**title**
Toggle version

**description**
Show the version number and exit

**`switch-clipboard`**

**title**
Switch clipboard

**description**
Switch the value of the “clipboard” configuration option.

**`set-clipboard-external`**

**title**
Set clipboard to external

**description**
Set the value of the “clipboard” configuration option to “external”

**`set-clipboard-internal`**

**title**
Set clipboard to internal

**description**
Set the value of the “clipboard” configuration option to “internal”

**`set-clipboard-terminal`**

**title**
Set clipboard to terminal

**description**
Set the value of the “clipboard” configuration option to “terminal”

**`switch-log-level`**

**title**
Switch the log level

**description**
Switch the value of the “log_level” configuration option.

**`set-log-level-debug`**

**title**
Set the log level to debug

**description**
Set the value of the “log_level” configuration option to “debug”

**`set-log-level-info`**

**title**
Set the log level to info

**description**
Set the value of the “log_level” configuration option to “info”

**`set-log-level-warning`**

**title**
Set the log level to warning

**description**
Set the value of the “log_level” configuration option to “warning”

**`set-log-level-error`**

**title**
Set the log level to error

**description**
Set the value of the “log_level” configuration option to “error”

**`set-log-level-critical`**

**title**
Set the log level to critical

**description**
Set the value of the “log_level” configuration option to “critical”

**`toggle-show-shadows`**

**title**
Toggle show shadows

**description**
Show or hide shadows under menus and dialogs

**`toggle-show-status-bar`**

**title**
Toggle status bar

**description**
Show the status bar

**`toggle-set-cursor-shape`**

**title**
Toggle set cursor shape

**description**
Whether to set the shape of the cursor depending on the editing mode

**`toggle-cursor-blink`**

**title**
Toggle cursor blink

**description**
Whether to blink the cursor

**`quit`**

**title**
Quit

**description**
Quit euporie.

**`close-tab`**

**title**
Close tab

**description**
Close the current tab.

**`next-tab`**

**title**
Next tab

**description**
Switch to the next tab.

**`previous-tab`**

**title**
Previous tab

**description**
Switch to the previous tab.

**`focus-next`**

**title**
Focus next

**description**
Focus the next control.

**`focus-previous`**

**title**
Focus previous

**description**
Focus the previous control.

**`clear-screen`**

**title**
Clear screen

**description**
Clear the screen.

**`switch-edit-mode`**

**title**
Switch Editor key bindings

**description**
Switch the value of the “edit_mode” configuration option.

**`set-edit-mode-micro`**

**title**
Set Editor key bindings to micro

**description**
Set the value of the “edit_mode” configuration option to “micro”

**`set-edit-mode-emacs`**

**title**
Set Editor key bindings to emacs

**description**
Set the value of the “edit_mode” configuration option to “emacs”

**`set-edit-mode-vi`**

**title**
Set Editor key bindings to vi

**description**
Set the value of the “edit_mode” configuration option to “vi”

**`switch-tab-size`**

**title**
Switch tab size

**description**
Switch the value of the “tab_size” configuration option.

**`set-syntax-theme-abap`**

**title**
Set syntax theme to abap

**description**
Set the value of the “syntax_theme” configuration option to “abap”

**`set-syntax-theme-algol`**

**title**
Set syntax theme to algol

**description**
Set the value of the “syntax_theme” configuration option to “algol”

**`set-syntax-theme-algol_nu`**

**title**
Set syntax theme to algol_nu

**description**
Set the value of the “syntax_theme” configuration option to “algol_nu”

**`set-syntax-theme-arduino`**

**title**
Set syntax theme to arduino

**description**
Set the value of the “syntax_theme” configuration option to “arduino”

**`set-syntax-theme-autumn`**

**title**
Set syntax theme to autumn

**description**
Set the value of the “syntax_theme” configuration option to “autumn”

**`set-syntax-theme-bw`**

**title**
Set syntax theme to bw

**description**
Set the value of the “syntax_theme” configuration option to “bw”

**`set-syntax-theme-borland`**

**title**
Set syntax theme to borland

**description**
Set the value of the “syntax_theme” configuration option to “borland”

**`set-syntax-theme-coffee`**

**title**
Set syntax theme to coffee

**description**
Set the value of the “syntax_theme” configuration option to “coffee”

**`set-syntax-theme-colorful`**

**title**
Set syntax theme to colorful

**description**
Set the value of the “syntax_theme” configuration option to “colorful”

**`set-syntax-theme-default`**

**title**
Set syntax theme to default

**description**
Set the value of the “syntax_theme” configuration option to “default”

**`set-syntax-theme-dracula`**

**title**
Set syntax theme to dracula

**description**
Set the value of the “syntax_theme” configuration option to “dracula”

**`set-syntax-theme-emacs`**

**title**
Set syntax theme to emacs

**description**
Set the value of the “syntax_theme” configuration option to “emacs”

**`set-syntax-theme-friendly_grayscale`**

**title**
Set syntax theme to friendly_grayscale

**description**
Set the value of the “syntax_theme” configuration option to “friendly_grayscale”

**`set-syntax-theme-friendly`**

**title**
Set syntax theme to friendly

**description**
Set the value of the “syntax_theme” configuration option to “friendly”

**`set-syntax-theme-fruity`**

**title**
Set syntax theme to fruity

**description**
Set the value of the “syntax_theme” configuration option to “fruity”

**`set-syntax-theme-github-dark`**

**title**
Set syntax theme to github-dark

**description**
Set the value of the “syntax_theme” configuration option to “github-dark”

**`set-syntax-theme-gruvbox-dark`**

**title**
Set syntax theme to gruvbox-dark

**description**
Set the value of the “syntax_theme” configuration option to “gruvbox-dark”

**`set-syntax-theme-gruvbox-light`**

**title**
Set syntax theme to gruvbox-light

**description**
Set the value of the “syntax_theme” configuration option to “gruvbox-light”

**`set-syntax-theme-igor`**

**title**
Set syntax theme to igor

**description**
Set the value of the “syntax_theme” configuration option to “igor”

**`set-syntax-theme-inkpot`**

**title**
Set syntax theme to inkpot

**description**
Set the value of the “syntax_theme” configuration option to “inkpot”

**`set-syntax-theme-lightbulb`**

**title**
Set syntax theme to lightbulb

**description**
Set the value of the “syntax_theme” configuration option to “lightbulb”

**`set-syntax-theme-lilypond`**

**title**
Set syntax theme to lilypond

**description**
Set the value of the “syntax_theme” configuration option to “lilypond”

**`set-syntax-theme-lovelace`**

**title**
Set syntax theme to lovelace

**description**
Set the value of the “syntax_theme” configuration option to “lovelace”

**`set-syntax-theme-manni`**

**title**
Set syntax theme to manni

**description**
Set the value of the “syntax_theme” configuration option to “manni”

**`set-syntax-theme-material`**

**title**
Set syntax theme to material

**description**
Set the value of the “syntax_theme” configuration option to “material”

**`set-syntax-theme-monokai`**

**title**
Set syntax theme to monokai

**description**
Set the value of the “syntax_theme” configuration option to “monokai”

**`set-syntax-theme-murphy`**

**title**
Set syntax theme to murphy

**description**
Set the value of the “syntax_theme” configuration option to “murphy”

**`set-syntax-theme-native`**

**title**
Set syntax theme to native

**description**
Set the value of the “syntax_theme” configuration option to “native”

**`set-syntax-theme-nord-darker`**

**title**
Set syntax theme to nord-darker

**description**
Set the value of the “syntax_theme” configuration option to “nord-darker”

**`set-syntax-theme-nord`**

**title**
Set syntax theme to nord

**description**
Set the value of the “syntax_theme” configuration option to “nord”

**`set-syntax-theme-one-dark`**

**title**
Set syntax theme to one-dark

**description**
Set the value of the “syntax_theme” configuration option to “one-dark”

**`set-syntax-theme-paraiso-dark`**

**title**
Set syntax theme to paraiso-dark

**description**
Set the value of the “syntax_theme” configuration option to “paraiso-dark”

**`set-syntax-theme-paraiso-light`**

**title**
Set syntax theme to paraiso-light

**description**
Set the value of the “syntax_theme” configuration option to “paraiso-light”

**`set-syntax-theme-pastie`**

**title**
Set syntax theme to pastie

**description**
Set the value of the “syntax_theme” configuration option to “pastie”

**`set-syntax-theme-perldoc`**

**title**
Set syntax theme to perldoc

**description**
Set the value of the “syntax_theme” configuration option to “perldoc”

**`set-syntax-theme-rainbow_dash`**

**title**
Set syntax theme to rainbow_dash

**description**
Set the value of the “syntax_theme” configuration option to “rainbow_dash”

**`set-syntax-theme-rrt`**

**title**
Set syntax theme to rrt

**description**
Set the value of the “syntax_theme” configuration option to “rrt”

**`set-syntax-theme-sas`**

**title**
Set syntax theme to sas

**description**
Set the value of the “syntax_theme” configuration option to “sas”

**`set-syntax-theme-solarized-dark`**

**title**
Set syntax theme to solarized-dark

**description**
Set the value of the “syntax_theme” configuration option to “solarized-dark”

**`set-syntax-theme-solarized-light`**

**title**
Set syntax theme to solarized-light

**description**
Set the value of the “syntax_theme” configuration option to “solarized-light”

**`set-syntax-theme-staroffice`**

**title**
Set syntax theme to staroffice

**description**
Set the value of the “syntax_theme” configuration option to “staroffice”

**`set-syntax-theme-stata-dark`**

**title**
Set syntax theme to stata-dark

**description**
Set the value of the “syntax_theme” configuration option to “stata-dark”

**`set-syntax-theme-stata-light`**

**title**
Set syntax theme to stata-light

**description**
Set the value of the “syntax_theme” configuration option to “stata-light”

**`set-syntax-theme-tango`**

**title**
Set syntax theme to tango

**description**
Set the value of the “syntax_theme” configuration option to “tango”

**`set-syntax-theme-trac`**

**title**
Set syntax theme to trac

**description**
Set the value of the “syntax_theme” configuration option to “trac”

**`set-syntax-theme-vim`**

**title**
Set syntax theme to vim

**description**
Set the value of the “syntax_theme” configuration option to “vim”

**`set-syntax-theme-vs`**

**title**
Set syntax theme to vs

**description**
Set the value of the “syntax_theme” configuration option to “vs”

**`set-syntax-theme-xcode`**

**title**
Set syntax theme to xcode

**description**
Set the value of the “syntax_theme” configuration option to “xcode”

**`set-syntax-theme-zenburn`**

**title**
Set syntax theme to zenburn

**description**
Set the value of the “syntax_theme” configuration option to “zenburn”

**`switch-color-depth`**

**title**
Switch color depth

**description**
Switch the value of the “color_depth” configuration option.

**`set-color-depth-1`**

**title**
Set color depth to 1

**description**
Set the value of the “color_depth” configuration option to “1”

**`set-color-depth-4`**

**title**
Set color depth to 4

**description**
Set the value of the “color_depth” configuration option to “4”

**`set-color-depth-8`**

**title**
Set color depth to 8

**description**
Set the value of the “color_depth” configuration option to “8”

**`set-color-depth-24`**

**title**
Set color depth to 24

**description**
Set the value of the “color_depth” configuration option to “24”

**`toggle-multiplexer-passthrough`**

**title**
Toggle multiplexer passthrough

**description**
Use passthrough from within terminal multiplexers

**`switch-color-scheme`**

**title**
Switch color scheme

**description**
Switch the value of the “color_scheme” configuration option.

**`set-color-scheme-default`**

**title**
Set color scheme to default

**description**
Set the value of the “color_scheme” configuration option to “default”

**`set-color-scheme-inverse`**

**title**
Set color scheme to inverse

**description**
Set the value of the “color_scheme” configuration option to “inverse”

**`set-color-scheme-light`**

**title**
Set color scheme to light

**description**
Set the value of the “color_scheme” configuration option to “light”

**`set-color-scheme-dark`**

**title**
Set color scheme to dark

**description**
Set the value of the “color_scheme” configuration option to “dark”

**`set-color-scheme-black`**

**title**
Set color scheme to black

**description**
Set the value of the “color_scheme” configuration option to “black”

**`set-color-scheme-white`**

**title**
Set color scheme to white

**description**
Set the value of the “color_scheme” configuration option to “white”

**`set-color-scheme-custom`**

**title**
Set color scheme to custom

**description**
Set the value of the “color_scheme” configuration option to “custom”

**`switch-graphics`**

**title**
Switch graphics

**description**
Switch the value of the “graphics” configuration option.

**`set-graphics-none`**

**title**
Set graphics to none

**description**
Set the value of the “graphics” configuration option to “none”

**`set-graphics-sixel`**

**title**
Set graphics to sixel

**description**
Set the value of the “graphics” configuration option to “sixel”

**`set-graphics-kitty`**

**title**
Set graphics to kitty

**description**
Set the value of the “graphics” configuration option to “kitty”

**`set-graphics-iterm`**

**title**
Set graphics to iterm

**description**
Set the value of the “graphics” configuration option to “iterm”

**`toggle-force-graphics`**

**title**
Toggle force graphics

**description**
Force use of specified graphics protocol

**`toggle-enable-language-servers`**

**title**
Toggle enable language servers

**description**
Enable language server support

**`switch-app`**

**title**
Switch app

**description**
Switch the value of the “app” configuration option.

**`set-app-notebook`**

**title**
Set app to notebook

**description**
Set the value of the “app” configuration option to “notebook”

**`set-app-console`**

**title**
Set app to console

**description**
Set the value of the “app” configuration option to “console”

**`switch-port`**

**title**
Switch port

**description**
Switch the value of the “port” configuration option.

**`toggle-auth`**

**title**
Toggle auth

**description**
Allow unauthenticated access to euporie hub

## 5.12 euporie

### Modules

| | |
|---|---|
| *euporie.console* | An interactive jupyter console with rich output. |
| *euporie.core* | This package defines the euporie application and its components. |
| *euporie.hub* | A multi-user hub euporie application. |
| *euporie.notebook* | Define euporie's application classes. |
| *euporie.preview* | A euporie app for previewing notebook files. |

### 5.12.1 euporie.console

An interactive jupyter console with rich output.

#### Modules

| | |
|---|---|
| *euporie.console.app* | A text base user interface for euporie. |
| *euporie.console.tabs* | Contain various application tab implementations. |

#### euporie.console.app

A text base user interface for euporie.

#### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *cast*(typ, val) | Cast a value to a type. |
| *get_app*() | Get the current application. |
| *in_terminal*([render_cli_done]) | Asynchronous context manager that suspends the current application and runs the body in the terminal. |
| *ptk_get_app*() | Get the current active (running) Application. |

#### euporie.console.app.add_cmd

euporie.console.app.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

#### euporie.console.app.add_setting

euporie.console.app.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.console.app.cast

euporie.console.app.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.console.app.get_app

euporie.console.app.**get_app**() → *ConsoleApp*

Get the current application.

### euporie.console.app.in_terminal

euporie.console.app.**in_terminal**(*render_cli_done: bool = False*) → AsyncGenerator[None, None]

Asynchronous context manager that suspends the current application and runs the body in the terminal.

```
async def f():
    async with in_terminal():
        call_some_function()
        await call_some_async_function()
```

### euporie.console.app.ptk_get_app

euporie.console.app.**ptk_get_app**() → *Application*[Any]

Get the current active (running) Application. An *Application* is active during the Application.run_async() call.

We assume that there can only be one *Application* active at the same time. There is only one terminal window, with only one stdin and stdout. This makes the code significantly easier than passing around the *Application* everywhere.

If no *Application* is running, then return by default a DummyApplication. For practical reasons, we prefer to not raise an exception. This way, we don't have to check all over the place whether an actual *Application* was returned.

(For applications like pymux where we can have more than one *Application*, we'll use a work-around to handle that.)

### Classes

| | |
|---|---|
| *AboutDialog*(app) | A dialog which shows an "about" message. |
| *BaseApp*([title, set_title, leave_graphics, ...]) | All euporie apps. |
| *CommandPalette*(app) | A command palette which allows searching the available commands. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *Console*(app[, path, use_kernel_history, ...]) | Interactive console. |
| *ConsoleApp*(**kwargs) | Conole app. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *DisableMouseOnScroll*(content) | A container which disables mouse support on unhandled scroll up events. |
| *FloatContainer*(content, floats[, modal, ...]) | Container which can contain another container for the background, as well as a list of floating containers on top of it. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. ::. |
| *NoKernelsDialog*(app) | Dialog to warn the user that no installed kernels were found. |
| *Pager*([height]) | Interactive help pager. |
| *SaveAsDialog*(app) | A dialog which prompts the user for a filepath to save the current tab. |
| *SearchBar*([search_buffer, vi_mode, ...]) | Search mode. |
| *SelectKernelDialog*(app) | A dialog which allows the user to select a kernel. |
| *ShortcutsDialog*(app) | Display details of registered key-bindings in a dialog. |
| *StatusBar*([extra_filter, default]) | A status bar which shows the status of the current tab. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. ::. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |

### euporie.console.app.AboutDialog

**class** euporie.console.app.**AboutDialog**(*app:* BaseApp)

A dialog which shows an “about” message.

### euporie.console.app.BaseApp

**class** euporie.console.app.**BaseApp**(*title: str | None = None*, *set_title: bool = True*, *leave_graphics: FilterOrBool = True*, *extend_renderer_height: FilterOrBool = False*, *extend_renderer_width: FilterOrBool = False*, *enable_page_navigation_bindings: FilterOrBool | None = True*, ***kwargs: Any*)

All euporie apps.

The base euporie application class.

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

### euporie.console.app.CommandPalette

**class** euporie.console.app.**CommandPalette**(*app:* BaseApp)

A command palette which allows searching the available commands.

### euporie.console.app.ConditionalContainer

**class** euporie.console.app.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.console.app.Console

**class** euporie.console.app.**Console**(*app:* BaseApp, *path:* Path | *None* = *None*, *use_kernel_history: bool* = *True*, *connection_file: str* = *''*)

Interactive console.

An interactive console which connects to a Jupyter kernel.

### euporie.console.app.ConsoleApp

**class** euporie.console.app.**ConsoleApp**(***kwargs: Any*)

Conole app.

An interactive console which connects to Jupyter kernels and displays rich output in the terminal.

### euporie.console.app.Dimension

**class** euporie.console.app.**Dimension**(*min: int* | *None* = *None*, *max: int* | *None* = *None*, *weight: int* | *None* = *None*, *preferred: int* | *None* = *None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.console.app.DisableMouseOnScroll

**class** euporie.console.app.**DisableMouseOnScroll**(*content: AnyContainer*)

A container which disables mouse support on unhandled scroll up events.

This enables the terminal scroll-back buffer to be scrolled if there is nothing in the application which is scrollable.

### euporie.console.app.FloatContainer

**class** euporie.console.app.**FloatContainer**(*content: AnyContainer*, *floats: list[Float]*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*, *z_index: int | None = None*)

Container which can contain another container for the background, as well as a list of floating containers on top of it.

Example Usage:

```
FloatContainer(content=Window(...),
               floats=[
                   Float(xcursor=True,
                         ycursor=True,
                         content=CompletionsMenu(...))
               ])
```

**Parameters**

**z_index** – (int or None) When specified, this can be used to bring element in front of floating elements. *None* means: inherit from parent. This is the z_index for the whole *Float* container as a whole.

### euporie.console.app.HSplit

**class** euporie.console.app.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

```
+--------------------+
|                    |
+--------------------+
|                    |
+--------------------+
```

By default, this doesn't display a horizontal line between the children, but if this is something you need, then create a HSplit as follows:

```
HSplit(children=[ ... ], padding_char='-',
       padding=1, padding_style='#ffff00')
```

**Parameters**

- **children** – List of child *Container* objects.
- **window_too_small** – A *Container* object that is displayed if there is not enough space for all the children. By default, this is a "Window too small" message.
- **align** – *VerticalAlign* value.
- **width** – When given, use this width instead of looking at the children.
- **height** – When given, use this height instead of looking at the children.
- **z_index** – (int or None) When specified, this can be used to bring element in front of floating elements. *None* means: inherit from parent.
- **style** – A style string.
- **modal** – `True` or `False`.
- **key_bindings** – `None` or a *KeyBindings* object.
- **padding** – (*Dimension* or int), size to be used for the padding.
- **padding_char** – Character to be used for filling in the padding.
- **padding_style** – Style to applied to the padding.

### euporie.console.app.NoKernelsDialog

**class** euporie.console.app.**NoKernelsDialog**(*app:* BaseApp)

Dialog to warn the user that no installed kernels were found.

### euporie.console.app.Pager

**class** euporie.console.app.**Pager**(*height: AnyDimension | None = None*)

Interactive help pager.

A pager which displays information at the bottom of a tab.

### euporie.console.app.SaveAsDialog

**class** euporie.console.app.**SaveAsDialog**(*app:* BaseApp)

A dialog which prompts the user for a filepath to save the current tab.

### euporie.console.app.SearchBar

**class** euporie.console.app.**SearchBar**(*search_buffer:* Buffer | *None = None*, *vi_mode: bool = False*, *text_if_not_searching: AnyFormattedText = ''*, *forward_search_prompt: AnyFormattedText = 'I-search: '*, *backward_search_prompt: AnyFormattedText = 'I-search backward: '*, *ignore_case: FilterOrBool = False*)

Search mode.

A search toolbar with custom style and text.

### euporie.console.app.SelectKernelDialog

**class** euporie.console.app.**SelectKernelDialog**(*app:* BaseApp)

A dialog which allows the user to select a kernel.

### euporie.console.app.ShortcutsDialog

**class** euporie.console.app.**ShortcutsDialog**(*app:* BaseApp)

Display details of registered key-bindings in a dialog.

### euporie.console.app.StatusBar

**class** euporie.console.app.**StatusBar**(*extra_filter: FilterOrBool = True*, *default: StatusBarFields | None = None*)

A status bar which shows the status of the current tab.

### euporie.console.app.VSplit

**class** euporie.console.app.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

By default, this doesn't display a vertical line between the children, but if this is something you need, then create a HSplit as follows:

```
VSplit(children=[ ... ], padding_char='|',
       padding=1, padding_style='#ffff00')
```

**Parameters**

- **children** – List of child *Container* objects.
- **window_too_small** – A *Container* object that is displayed if there is not enough space for all the children. By default, this is a "Window too small" message.
- **align** – *HorizontalAlign* value.
- **width** – When given, use this width instead of looking at the children.
- **height** – When given, use this height instead of looking at the children.
- **z_index** – (int or None) When specified, this can be used to bring element in front of floating elements. *None* means: inherit from parent.

- **style** – A style string.
- **modal** – `True` or `False`.
- **key_bindings** – `None` or a *KeyBindings* object.
- **padding** – (*Dimension* or int), size to be used for the padding.
- **padding_char** – Character to be used for filling in the padding.
- **padding_style** – Style to applied to the padding.

### euporie.console.app.Window

**class** euporie.console.app.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

**Parameters**

- **content** – *UIControl* instance.
- **width** – *Dimension* instance or callable.
- **height** – *Dimension* instance or callable.
- **z_index** – When specified, this can be used to bring element in front of floating elements.
- **dont_extend_width** – When *True*, don't take up more width then the preferred width reported by the control.
- **dont_extend_height** – When *True*, don't take up more width then the preferred height reported by the control.
- **ignore_content_width** – A *bool* or *Filter* instance. Ignore the *UIContent* width when calculating the dimensions.
- **ignore_content_height** – A *bool* or *Filter* instance. Ignore the *UIContent* height when calculating the dimensions.
- **left_margins** – A list of *Margin* instance to be displayed on the left. For instance: `NumberedMargin` can be one of them in order to show line numbers.
- **right_margins** – Like *left_margins*, but on the other side.

- **scroll_offsets** – *ScrollOffsets* instance, representing the preferred amount of lines/columns to be always visible before/after the cursor. When both top and bottom are a very high number, the cursor will be centered vertically most of the time.
- **allow_scroll_beyond_bottom** – A *bool* or *Filter* instance. When True, allow scrolling so far, that the top part of the content is not visible anymore, while there is still empty space available at the bottom of the window. In the Vi editor for instance, this is possible. You will see tildes while the top part of the body is hidden.
- **wrap_lines** – A *bool* or *Filter* instance. When True, don't scroll horizontally, but wrap lines instead.
- **get_vertical_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll. (When this is *None*, the scroll is only determined by the last and current cursor position.)
- **get_horizontal_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll.
- **always_hide_cursor** – A *bool* or *Filter* instance. When True, never display the cursor, even when the user control specifies a cursor position.
- **cursorline** – A *bool* or *Filter* instance. When True, display a cursorline.
- **cursorcolumn** – A *bool* or *Filter* instance. When True, display a cursorcolumn.
- **colorcolumns** – A list of `ColorColumn` instances that describe the columns to be highlighted, or a callable that returns such a list.
- **align** – *WindowAlign* value or callable that returns an *WindowAlign* value. alignment of content.
- **style** – A style string. Style to be applied to all the cells in this window. (This can be a callable that returns a string.)
- **char** – (string) Character to be used for filling the background. This can also be a callable that returns a character.
- **get_line_prefix** – None or a callable that returns formatted text to be inserted before a line. It takes a line number (int) and a wrap_count and returns formatted text. This can be used for implementation of line continuations, things like Vim "breakindent" and so on.

**class** euporie.console.app.**ConsoleApp**(***kwargs: Any*)

Bases: *BaseApp*

Conole app.

An interactive console which connects to Jupyter kernels and displays rich output in the terminal.

**async cancel_and_wait_for_background_tasks**() → None

Cancel all background tasks, and wait for the cancellation to complete. 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.)

**cleanup**(*signum: int*, *frame: FrameType | None*) → None

Restore the state of the terminal on unexpected exit.

**cleanup_closed_tab**(*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:* Tab | *None* = *None*) → None

Close a notebook tab.

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

**property 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.

**color_palette: *ColorPalette***

**cpr_not_supported_callback**() → None

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

**create_background_task**(*coroutine: Coroutine[Any, Any, None]*) → Task[None]

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

Given that we still support Python versions before 3.11, we can't use task groups (and exception groups), because of that, these background tasks are not allowed to raise exceptions. If they do, we'll call the default exception handler from the event loop.

If at some point, we have Python 3.11 as the minimum supported Python version, then we can use a *TaskGroup* (with the lifetime of *Application.run_async()*, and run run the background tasks in there.

This is not threadsafe.

**create_merged_style**() → *BaseStyle*

Generate a new merged style for the application.

Using a dynamic style has serious performance issues, so instead we update the style on the renderer directly when it changes in *self.update_style*

**Returns**
Return a combined style to use for the application

**property current_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: SearchState**

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

**draw**(*render_as_done: bool* = *True*) → None

Draw the app without focus, leaving the cursor below the drawn output.

**exit**(*result: _AppResult* | *None* = *None*, *exception: BaseException* | *type[BaseException]* | *None* = *None*, *style: str* = '') → None

Close all tabs on exit.

**focus_tab**(*tab:* Tab) → None

Make a tab visible and focuses it.

**get_edit_mode**() → *EditingMode*

Return the editing mode enum defined in the configuration.

**get_file_tab**(*path:* Path) → type[*Tab*] | None

Return the tab to use for a file path.

**get_file_tabs**(*path:* Path) → list[type[*Tab*]]

Return the tab to use for a file path.

**get_language_lsps**(*language: str*) → list[*euporie.core.lsp.LspClient*]

Return the approprrate LSP clients for a given language.

**get_used_style_strings**() → list[str]

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

**async classmethod interact**(*ssh_session:* PromptToolkitSSHSession) → None

Run the app asynchronously for the hub SSH server.

**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

Launch the app.

**load_container**() → *FloatContainer*

Return a container with all opened tabs.

**classmethod load_input**() → Input

Create the input for this application to use.

Ensures the TUI app always tries to run in a TTY.

**Returns**
A prompt-toolkit input instance

**load_key_bindings**() → None

Load the application's key bindings.

**classmethod load_output**() → Output

Create the output for this application to use.

Ensures the TUI app always tries to run in a TTY.

**Returns**
A prompt-toolkit output instance

**log_stdout_level**: str = 'ERROR'

**mouse_position**: *Point*

**name**: str = 'console'

**open_file**(*path:* Path, *read_only:* *bool* = *False*, *tab_class:* *type*[Tab] | *None* = *None*) → None

Create 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
- **tab_class** – The tab type to use to open the file

**open_files**() → None

Open the files defined in the configuration.

**pause_rendering**() → None

Block rendering, but allows input to be processed.

The first line prevents the display being drawn, and the second line means the key processor continues to process keys. We need this as we need to wait for the results of terminal queries which come in as key events.

This is used to prevent flicker when we update the styles based on terminal feedback.

**post_load**() → None

Allow subclasses to define additional loading steps.

**pre_run**(*app:* prompt_toolkit.application.application.Application | *None* = *None*) → None

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

**print_text**(*text: AnyFormattedText*, *style:* BaseStyle | *None* = *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.

**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.

**resume_rendering**() → None

Reume rendering the app.

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

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

This will run the application in a fresh asyncio event loop.

**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_appp().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: Callable[[], None] | None = None*, *set_exception_handler: bool = True*, *handle_sigint: bool = True*, *slow_callback_duration: float = 0.5*) → _AppResult

Run the application.

**async run_system_command**(*command: str*, *wait_for_enter: bool = True*, *display_before_text: AnyFormattedText = ''*, *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.

**shutdown_lsps**() → None

Shut down all the remaining LSP servers.

**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 syntax_theme: str**

Calculate the current syntax theme.

**property tab: Tab | None**

Return the currently selected tab container object.

**property tab_idx: int**

Get 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 uses 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.

**property title: str**

The application's title.

**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_edit_mode**(*setting:* Setting | *None* = *None*) → None

Set the keybindings for editing mode.

**update_style**(*query:* TerminalQuery | Setting | *None* = *None*) → None

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

**vi_state**

Vi state. (For Vi key bindings.)

euporie.console.app.**get_app**() → *ConsoleApp*

Get the current application.

### euporie.console.tabs

Contain various application tab implementations.

### Modules

| | |
|---|---|
| *euporie.console.tabs.console* | Contain the main class for a notebook file. |

### euporie.console.tabs.console

Contain the main class for a notebook file.

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *cast*(typ, val) | Cast a value to a type. |
| *edit_in_editor*(filename[, line_number]) | Suspend the current app and edit a file in an external editor. |
| *get_cmd*(name) | Get a command from the centralized command system by name. |
| *has_focus*(value) | Enable when this buffer has the focus. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *register_bindings*(bindings) | Update the key-binding registry. |

#### euporie.console.tabs.console.add_cmd

euporie.console.tabs.console.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

#### euporie.console.tabs.console.add_setting

euporie.console.tabs.console.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

#### euporie.console.tabs.console.cast

euporie.console.tabs.console.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

#### euporie.console.tabs.console.edit_in_editor

euporie.console.tabs.console.**edit_in_editor**(*filename: str*, *line_number: int = 0*) → None

Suspend the current app and edit a file in an external editor.

### euporie.console.tabs.console.get_cmd

euporie.console.tabs.console.**get_cmd**(*name: str*) → *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.console.tabs.console.has_focus

euporie.console.tabs.console.**has_focus**(*value: FocusableElement*) → *Condition*

Enable when this buffer has the focus.

### euporie.console.tabs.console.load_registered_bindings

euporie.console.tabs.console.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.console.tabs.console.register_bindings

euporie.console.tabs.console.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### Classes

| | |
|---|---|
| *Buffer*([completer, auto_suggest, history, ...]) | The core data structure that holds the text and cursor position of the current input line and implements all text manipulations on top of it. |
| *CellOutputArea*(json, parent[, style]) | An area below a cell where one or more cell outputs can be shown. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *Console*(app[, path, use_kernel_history, ...]) | Interactive console. |
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *FloatContainer*(content, floats[, modal, ...]) | Container which can contain another container for the background, as well as a list of floating containers on top of it. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. ::. |
| *KernelInput*(kernel_tab[, text, multiline, ...]) | Kernel input text areas. |
| *KernelTab*(app[, path, kernel, comms, ...]) | A Tab which connects to a kernel. |
| *KernelValidator*(kernel) | Validate kernel input using a kernel code completeness call. |
| *KeyBindings*() | A container for a set of key bindings. |
| *Layout*(container[, focused_element]) | The layout for a prompt_toolkit Application. |
| *LspCell*(id, idx, path, kind, language, text, ...) | An LSP client's representation of a cell. |
| *LspFormatter*(lsp, path[, languages]) | Format a document using a LSP server. |
| *MsgCallbacks* | Typed dictionary for named message callbacks. |
| *PrintingContainer*(children[, width, ...]) | A container which displays all it's children in a vertical list. |
| *Report*([iterable]) | Class for storing a diagnostic report. |
| *StdInput*(kernel_tab) | A widget to accept kernel input. |
| *UPath*(*args[, protocol]) | |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. ::. |
| *ValidationState*(value[, names, module, ...]) | The validation state of a buffer. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.console.tabs.console.Buffer

**class** euporie.console.tabs.console.**Buffer**(*completer:* prompt_toolkit.completion.base.Completer | *None* = *None*, *auto_suggest:* prompt_toolkit.auto_suggest.AutoSuggest | *None* = *None*, *history:* prompt_toolkit.history.History | *None* = *None*, *validator:* prompt_toolkit.validation.Validator | *None* = *None*, *tempfile_suffix: Union[str, Callable[[], str]]* = '', *tempfile: Union[str, Callable[[], str]]* = '', *name: str* = '', *complete_while_typing: Union[*Filter, *bool]* = *False*, *validate_while_typing: Union[*Filter, *bool]* = *False*, *enable_history_search: Union[*Filter, *bool]* = *False*, *document:* prompt_toolkit.document.Document | *None* = *None*, *accept_handler: Optional[Callable[[*Buffer*], bool]]* = *None*, *read_only: Union[*Filter, *bool]* = *False*, *multiline: Union[*Filter, *bool]* = *True*, *on_text_changed: Optional[Callable[[*Buffer*], None]]* = *None*, *on_text_insert: Optional[Callable[[*Buffer*], None]]* = *None*, *on_cursor_position_changed: Optional[Callable[[*Buffer*], None]]* = *None*, *on_completions_changed: Optional[Callable[[*Buffer*], None]]* = *None*, *on_suggestion_set: Optional[Callable[[*Buffer*], None]]* = *None*)

The core data structure that holds the text and cursor position of the current input line and implements all text manipulations on top of it. It also implements the history, undo stack and the completion state.

**Parameters**

- **completer** – `Completer` instance.
- **history** – *`History`* instance.
- **tempfile_suffix** – The tempfile suffix (extension) to be used for the "open in editor" function. For a Python REPL, this would be ".py", so that the editor knows the syntax highlighting to use. This can also be a callable that returns a string.
- **tempfile** – For more advanced tempfile situations where you need control over the subdirectories and filename. For a Git Commit Message, this would be ".git/COMMIT_EDITMSG", so that the editor knows the syntax highlighting to use. This can also be a callable that returns a string.
- **name** – Name for this buffer. E.g. DEFAULT_BUFFER. This is mostly useful for key bindings where we sometimes prefer to refer to a buffer by their name instead of by reference.
- **accept_handler** – Called when the buffer input is accepted. (Usually when the user presses *enter*.) The accept handler receives this *Buffer* as input and should return True when the buffer text should be kept instead of calling reset.

  In case of a *PromptSession* for instance, we want to keep the text, because we will exit the application, and only reset it during the next run.

Events:

**Parameters**

- **on_text_changed** – When the buffer text changes. (Callable or None.)
- **on_text_insert** – When new text is inserted. (Callable or None.)
- **on_cursor_position_changed** – When the cursor moves. (Callable or None.)
- **on_completions_changed** – When the completions were changed. (Callable or None.)

- **on_suggestion_set** – When an auto-suggestion text has been set. (Callable or None.)

Filters:

**Parameters**

- **complete_while_typing** – `Filter` or *bool*. Decide whether or not to do asynchronous autocompleting while typing.
- **validate_while_typing** – `Filter` or *bool*. Decide whether or not to do asynchronous validation while typing.
- **enable_history_search** – `Filter` or *bool* to indicate when up-arrow partial string matching is enabled. It is advised to not enable this at the same time as *complete_while_typing*, because when there is an autocompletion found, the up arrows usually browse through the completions, rather than through the history.
- **read_only** – `Filter`. When True, changes will not be allowed.
- **multiline** – `Filter` or *bool*. When not set, pressing *Enter* will call the *accept_handler*. Otherwise, pressing *Esc-Enter* is required.

### euporie.console.tabs.console.CellOutputArea

**class** euporie.console.tabs.console.**CellOutputArea**(*json: list[dict[str, Any]]*, *parent: OutputParent | None*, *style: str = ''*)

An area below a cell where one or more cell outputs can be shown.

### euporie.console.tabs.console.Condition

**class** euporie.console.tabs.console.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.console.tabs.console.ConditionalContainer

**class** euporie.console.tabs.console.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.console.tabs.console.Console

**class** euporie.console.tabs.console.**Console**(*app:* BaseApp, *path:* Path | *None* = *None*, *use_kernel_history:* *bool* = *True*, *connection_file:* *str* = *''*)

Interactive console.

An interactive console which connects to a Jupyter kernel.

### euporie.console.tabs.console.Event

**class** euporie.console.tabs.console.**Event**(*sender: _Sender*, *handler: Optional[Callable[[_Sender], None]]* = *None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.console.tabs.console.FloatContainer

**class** euporie.console.tabs.console.**FloatContainer**(*content: AnyContainer*, *floats: list[*Float*]*, *modal:* *bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style:* *str* | *Callable[[], str]* = *''*, *z_index:* *int* | *None* = *None*)

Container which can contain another container for the background, as well as a list of floating containers on top of it.

Example Usage:

```
FloatContainer(content=Window(...),
               floats=[
                   Float(xcursor=True,
                         ycursor=True,
                         content=CompletionsMenu(...))
               ])
```

**Parameters**

**z_index** – (int or None) When specified, this can be used to bring element in front of floating

elements. *None* means: inherit from parent. This is the z_index for the whole *Float* container as a whole.

### euporie.console.tabs.console.FormattedTextControl

**class** euporie.console.tabs.console.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **key_bindings** – a *KeyBindings* object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.console.tabs.console.HSplit

**class** euporie.console.tabs.console.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

```
+--------------------+
|                    |
+--------------------+
|                    |
+--------------------+
```

By default, this doesn't display a horizontal line between the children, but if this is something you need, then create a HSplit as follows:

```
HSplit(children=[ ... ], padding_char='-',
       padding=1, padding_style='#ffff00')
```

**Parameters**

- **children** – List of child *Container* objects.
- **window_too_small** – A *Container* object that is displayed if there is not enough space for all the children. By default, this is a "Window too small" message.
- **align** – *VerticalAlign* value.
- **width** – When given, use this width instead of looking at the children.
- **height** – When given, use this height instead of looking at the children.
- **z_index** – (int or None) When specified, this can be used to bring element in front of floating elements. *None* means: inherit from parent.
- **style** – A style string.
- **modal** – `True` or `False`.
- **key_bindings** – `None` or a *KeyBindings* object.
- **padding** – (*Dimension* or int), size to be used for the padding.
- **padding_char** – Character to be used for filling in the padding.
- **padding_style** – Style to applied to the padding.

### euporie.console.tabs.console.KernelInput

**class** euporie.console.tabs.console.**KernelInput**(*kernel_tab: KernelTab*, *text: str = ''*, *multiline: FilterOrBool = True*, *password: FilterOrBool = False*, *lexer: Lexer | None = None*, *auto_suggest: AutoSuggest | None = None*, *completer: Completer | None = None*, *complete_while_typing: FilterOrBool = True*, *validator: Validator | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *history: History | None = None*, *focusable: FilterOrBool = True*, *focus_on_click: FilterOrBool = True*, *wrap_lines: FilterOrBool = False*, *read_only: FilterOrBool = False*, *width: AnyDimension = None*, *height: AnyDimension = None*, *dont_extend_height: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *line_numbers: bool = False*, *get_line_prefix: GetLinePrefixCallable | None = None*, *scrollbar: FilterOrBool = True*, *style: str = 'class:kernel-input'*, *search_field: SearchToolbar | None = None*, *preview_search: FilterOrBool = False*, *prompt: AnyFormattedText = ''*, *input_processors: list[Processor] | None = None*, *name: str = ''*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *on_cursor_position_changed: Callable[[Buffer], None] | None = None*, *tempfile_suffix: str | Callable[[], str] = ''*, *key_bindings: KeyBindingsBase | None = None*, *enable_history_search: FilterOrBool = False*, *autosuggest_while_typing: FilterOrBool = True*, *validate_while_typing: FilterOrBool = False*, *scroll_offsets: ScrollOffsets | None = None*, *formatters: list[Formatter] | None = None*, *language: str | Callable[[], str] | None = None*, *diagnostics: Report | Callable[[], Report] | None = None*, *inspector: Inspector | None = None*, *show_diagnostics: FilterOrBool = True*)

Kernel input text areas.

A customized text area for the cell input.

### euporie.console.tabs.console.KernelTab

**class** euporie.console.tabs.console.**KernelTab**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *False*, *connection_file:* Path | *None* = *None*)

A Tab which connects to a kernel.

### euporie.console.tabs.console.KernelValidator

**class** euporie.console.tabs.console.**KernelValidator**(*kernel:* Kernel)

Validate kernel input using a kernel code completeness call.

### euporie.console.tabs.console.KeyBindings

**class** euporie.console.tabs.console.**KeyBindings**

A container for a set of key bindings.

Example usage:

```
kb = KeyBindings()

@kb.add('c-t')
def _(event):
    print('Control-T pressed')

@kb.add('c-a', 'c-b')
def _(event):
    print('Control-A pressed, followed by Control-B')

@kb.add('c-x', filter=is_searching)
def _(event):
    print('Control-X pressed')  # Works only if we are searching.
```

### euporie.console.tabs.console.Layout

**class** euporie.console.tabs.console.**Layout**(*container: AnyContainer*, *focused_element:* *FocusableElement* | *None* = *None*)

The layout for a prompt_toolkit `Application`. This also keeps track of which user control is focused.

**Parameters**

- **container** – The “root” container for the layout.
- **focused_element** – element to be focused initially. (Can be anything the *focus* function accepts.)

### euporie.console.tabs.console.LspCell

**class** euporie.console.tabs.console.**LspCell**(*id: str*, *idx: int*, *path:* Path, *kind: str*, *language: str*, *text: str*, *execution_count: int*, *metadata: dict[str, Any] | None = None*)

An LSP client's representation of a cell.

### euporie.console.tabs.console.LspFormatter

**class** euporie.console.tabs.console.**LspFormatter**(*lsp:* LspClient, *path:* Path, *languages: set[str] | None = None*)

Format a document using a LSP server.

### euporie.console.tabs.console.MsgCallbacks

**class** euporie.console.tabs.console.**MsgCallbacks**

Typed dictionary for named message callbacks.

### euporie.console.tabs.console.PrintingContainer

**class** euporie.console.tabs.console.**PrintingContainer**(*children: Callable | Sequence[AnyContainer]*, *width: AnyDimension = None*, *key_bindings:* KeyBindingsBase *| None = None*)

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

### euporie.console.tabs.console.Report

**class** euporie.console.tabs.console.**Report**(*iterable=()*, */*)

Class for storing a diagnostic report.

### euporie.console.tabs.console.StdInput

**class** euporie.console.tabs.console.**StdInput**(*kernel_tab:* KernelTab)

A widget to accept kernel input.

### euporie.console.tabs.console.UPath

**class** euporie.console.tabs.console.**UPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

### euporie.console.tabs.console.VSplit

**class** euporie.console.tabs.console.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

By default, this doesn't display a vertical line between the children, but if this is something you need, then create a HSplit as follows:

```
VSplit(children=[ ... ], padding_char='|',
       padding=1, padding_style='#ffff00')
```

**Parameters**

- **children** – List of child *Container* objects.
- **window_too_small** – A *Container* object that is displayed if there is not enough space for all the children. By default, this is a "Window too small" message.
- **align** – *HorizontalAlign* value.
- **width** – When given, use this width instead of looking at the children.
- **height** – When given, use this height instead of looking at the children.
- **z_index** – (int or None) When specified, this can be used to bring element in front of floating elements. *None* means: inherit from parent.
- **style** – A style string.
- **modal** – `True` or `False`.
- **key_bindings** – `None` or a *KeyBindings* object.
- **padding** – (*Dimension* or int), size to be used for the padding.
- **padding_char** – Character to be used for filling in the padding.
- **padding_style** – Style to applied to the padding.

### euporie.console.tabs.console.ValidationState

**class** euporie.console.tabs.console.**ValidationState**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

The validation state of a buffer. This is set after the validation.

### euporie.console.tabs.console.Window

**class** euporie.console.tabs.console.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

**Parameters**

- **content** – *UIControl* instance.
- **width** – *Dimension* instance or callable.
- **height** – *Dimension* instance or callable.
- **z_index** – When specified, this can be used to bring element in front of floating elements.
- **dont_extend_width** – When *True*, don't take up more width then the preferred width reported by the control.
- **dont_extend_height** – When *True*, don't take up more width then the preferred height reported by the control.
- **ignore_content_width** – A *bool* or *Filter* instance. Ignore the *UIContent* width when calculating the dimensions.
- **ignore_content_height** – A *bool* or *Filter* instance. Ignore the *UIContent* height when calculating the dimensions.
- **left_margins** – A list of *Margin* instance to be displayed on the left. For instance: NumberedMargin can be one of them in order to show line numbers.

- **right_margins** – Like *left_margins*, but on the other side.
- **scroll_offsets** – *ScrollOffsets* instance, representing the preferred amount of lines/columns to be always visible before/after the cursor. When both top and bottom are a very high number, the cursor will be centered vertically most of the time.
- **allow_scroll_beyond_bottom** – A *bool* or *Filter* instance. When True, allow scrolling so far, that the top part of the content is not visible anymore, while there is still empty space available at the bottom of the window. In the Vi editor for instance, this is possible. You will see tildes while the top part of the body is hidden.
- **wrap_lines** – A *bool* or *Filter* instance. When True, don't scroll horizontally, but wrap lines instead.
- **get_vertical_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll. (When this is *None*, the scroll is only determined by the last and current cursor position.)
- **get_horizontal_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll.
- **always_hide_cursor** – A *bool* or *Filter* instance. When True, never display the cursor, even when the user control specifies a cursor position.
- **cursorline** – A *bool* or *Filter* instance. When True, display a cursorline.
- **cursorcolumn** – A *bool* or *Filter* instance. When True, display a cursorcolumn.
- **colorcolumns** – A list of ColorColumn instances that describe the columns to be highlighted, or a callable that returns such a list.
- **align** – *WindowAlign* value or callable that returns an *WindowAlign* value. alignment of content.
- **style** – A style string. Style to be applied to all the cells in this window. (This can be a callable that returns a string.)
- **char** – (string) Character to be used for filling the background. This can also be a callable that returns a character.
- **get_line_prefix** – None or a callable that returns formatted text to be inserted before a line. It takes a line number (int) and a wrap_count and returns formatted text. This can be used for implementation of line continuations, things like Vim "breakindent" and so on.

### euporie.console.tabs.console.partial

**class** euporie.console.tabs.console.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.console.tabs.console.**Console**(*app:* BaseApp, *path:* Path | *None* = *None*, *use_kernel_history:* *bool* = *True*, *connection_file:* *str* = '')

Bases: *KernelTab*

Interactive console.

An interactive console which connects to a Jupyter kernel.

**accept_stdin**(*buf:* Buffer) → bool

Accept the user's input.

**allow_stdin: bool**

**bg_init = False**

**change_kernel**(*msg: str | None = None*, *startup: bool = False*) → None
Prompt the user to select a new kernel.

**clear_output**(*wait: bool = False*) → None
Remove the last output, optionally when new output is generated.

**close**(*cb: Callable | None = None*) → None
Close the console tab.

**comm_close**(*content: dict*, *buffers: Sequence[bytes]*) → None
Close a notebook Comm.

**comm_msg**(*content: dict*, *buffers: Sequence[bytes]*) → None
Respond to a Comm message from the kernel.

**comm_open**(*content: dict*, *buffers: Sequence[bytes]*) → None
Register a new kernel Comm object in the notebook.

**complete**(*content: dict | None = None*) → None
Re-render any changes.

**container: AnyContainer**

**property current_input: *KernelInput***
Return the currently active kernel input, if any.

**default_callbacks: *MsgCallbacks***

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None
Focus the tab (or make it visible).

**init_kernel**(*kernel: Kernel | None = None*, *comms: dict[str, Comm] | None = None*, *use_kernel_history: bool = False*, *connection_file: Path | None = None*) → None
Set up the tab's kernel and related components.

**interrupt_kernel**() → None
Interrupt the current *Notebook*'s kernel.

**kernel: *Kernel***

**kernel_died**() → None
Call when the kernel dies.

**property kernel_display_name: str**
Return the display name of the kernel defined in the notebook JSON.

**property kernel_lang_file_ext: str**
Return the display name of the kernel defined in the notebook JSON.

**kernel_language: str**

**property kernel_name: str**
Return the name of the kernel defined in the notebook JSON.

**kernel_started**(*result: dict[str, Any] | None = None*) → None
Task to run when the kernel has started.

**lang_file_ext**() → str
Return the file extension for scripts in the notebook's language.

**property language: str**
The language of the current kernel.

**load_container**() → *HSplit*
Build the main application layout.

**async load_history**() → None
Load kernel history.

**async load_lsps**() → None
Load the LSP clients.

**lsp_add_cell**(*lsp:* LspClient) → None
Notify the LSP of a new cell.

**lsp_after_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we saved a document.

**lsp_before_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we are about to save a document.

**property lsp_cell: *LspCell***
Return a LSP cell representation of the current input.

**lsp_change_handler**(*lsp:* LspClient) → None
Tell the LSP server a file has changed.

**lsp_close_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_open_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_update_diagnostics**(*lsp:* LspClient) → None
Process a new diagnostic report from the LSP.

**property metadata: dict[str, Any]**
Return a dictionary to hold notebook / kernel metadata.

**mime_types: ClassVar[set[str]] = {}**

**name: str | None = None**

**new_output**(*output_json: dict[str, Any]*) → None
Print the previous output and replace it with the new one.

**property path_cell: *Path***
Return the virtual path of the console as a notebook cell.

**property path_nb: *Path***
Return the virtual path of the console as a notebook.

**post_init_kernel**() → None
Run stuff after the kernel is loaded.

**pre_init_kernel**() → None
Run stuff before the kernel is loaded.

**prompt**(*text: str*, *offset: int = 0*, *show_busy: bool = False*) → StyleAndTextTuples
Determine what should be displayed in the prompt of the cell.

**refresh**(*now: bool = True*) → None
Request the output is refreshed (refresh the whole app).

**render_outputs**(*app:* Application*[Any]*) → None
Request that any unrendered outputs be rendered.

**report**() → *Report*
Return the current diagnostic reports.

**report_kernel_error**(*error: Exception | None*) → None
Report a kernel error to the user.

**reset**() → None
Reset the state of the tab.

**restart_kernel**(*cb: Callable | None = None*) → None
Restart the current *Notebook*'s kernel.

**run**(*buffer:* prompt_toolkit.buffer.Buffer *| None = None*) → None
Run the code in the input box.

**save**(*path:* Path *| None = None*, *cb: Callable | None = None*) → None
Save the console as a notebook.

**set_kernel_info**(*info: dict*) → None
Receive and processes kernel metadata.

**set_next_input**(*text: str*, *replace: bool = False*) → None
Set the text for the next prompt.

**property title: str**
Return the tab title.

**validate_input**(*code: str*) → bool
Determine if the entered code is ready to run.

**weight: int = 0**

### 5.12.2 euporie.core

This package defines the euporie application and its components.

## Modules

| | |
|---|---|
| euporie.core.app | Contain the main Application class which runs euporie.core. |
| euporie.core.border | Define border styles. |
| euporie.core.clipboard | Module concerning clipboard access and manipulation. |
| euporie.core.comm | Sub-module for handling kernel Comm messages. |
| euporie.core.commands | Define a command object for use in key-bindings, menus, and the command palette. |
| euporie.core.completion | Contain the main class for a notebook file. |
| euporie.core.config | Define a configuration class for euporie.core. |
| euporie.core.convert | Sub-module concerned with the conversion of data formats. |
| euporie.core.current | Allow access to the current running application. |
| euporie.core.data_structures | Contain commonly used data structures. |
| euporie.core.diagnostics | Contains container classes for diagnostic information. |
| euporie.core.filters | Define common filters. |
| euporie.core.format | Contain functions to automatically format code cell input. |
| euporie.core.ft | Contain modules for working with formatted text. |
| euporie.core.graphics | Define controls for display of terminal graphics. |
| euporie.core.history | Define input history loaders. |
| euporie.core.inspection | Show contextual help for the current item under the cursor. |
| euporie.core.io | Define custom inputs and outputs, and related methods. |
| euporie.core.kernel | Contain the main class for a notebook file. |
| euporie.core.key_binding | Define key-bindings for the application. |
| euporie.core.keys | Register additional key escape sequences. |
| euporie.core.launch | Define a simple app for launching euporie apps. |
| euporie.core.layout | Contains containers and controls relating to layout. |
| euporie.core.lexers | Relating to lexers. |
| euporie.core.log | Initiate logging for euporie.core. |
| euporie.core.lsp | Defines a simple LSP client. |
| euporie.core.margins | Contain margins. |
| euporie.core.path | Responsible for loading data from urls. |
| euporie.core.processors | Buffer processors. |
| euporie.core.pygments | Contain lexers for pygments. |
| euporie.core.reference | Contain data reference dictionaries for value lookups. |
| euporie.core.renderer | Extended version of prompt_toolkit's renderer. |
| euporie.core.style | Style related functions. |
| euporie.core.suggest | Suggest line completions from kernel history. |
| euporie.core.tabs | Contain various application tab implementations. |
| euporie.core.terminal | Contain classes related to querying terminal features. |
| euporie.core.utils | Miscellaneou utility classes. |
| euporie.core.validation | Custom validators. |
| euporie.core.widgets | Contain various widgets used in euporie.core. |

### euporie.core.app

Contain the main Application class which runs euporie.core.

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *build_style*(cp[, have_term_colors]) | Create an application style based on the given color palette. |
| *cast*(typ, val) | Cast a value to a type. |
| *create_app_session*([input, output]) | Create a separate AppSession. |
| *create_input*([stdin, always_prefer_tty]) | Create the appropriate *Input* object for the current os/environment. |
| *create_output*([stdout, always_prefer_tty]) | Return an `Output` instance for the command line. |
| *get_app*() | Get the current active (running) Application. |
| *get_mime*(path) | Attempt to determine the mime-type of a path. |
| *get_style_by_name*(name) | Return a style class by its short name. |
| *load_cpr_bindings*() | |
| *load_emacs_bindings*() | Some e-macs extensions. |
| *load_emacs_search_bindings*() | |
| *load_emacs_shift_selection_bindings*() | Bindings to select text with shift + cursor movements |
| *load_ptk_basic_bindings*() | |
| *load_ptk_mouse_bindings*() | Key bindings, required for mouse support. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *load_vi_bindings*() | Vi extensions. |
| *load_vi_search_bindings*() | |
| *merge_key_bindings*(bindings) | Merge multiple `Keybinding` objects together. |
| *merge_style_transformations*(...) | Merge multiple transformations together. |
| *merge_styles*(styles) | Merge multiple *Style* objects. |
| *parse_path*(path[, resolve]) | Parse and resolve a path. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *set_app*(app) | Context manager that sets the given *Application* active in an *AppSession*. |
| *setup_logs*([config]) | Configure the logger for euporie. |
| *style_from_pygments_cls*(pygments_style_cls) | Shortcut to create a *Style* instance from a Pygments style class and a style dictionary. |
| *to_container*(container) | Make sure that the given object is a *Container*. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |

### euporie.core.app.add_cmd

euporie.core.app.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.app.add_setting

euporie.core.app.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.app.build_style

euporie.core.app.**build_style**(*cp:* ColorPalette, *have_term_colors: bool = True*) → *Style*

Create an application style based on the given color palette.

### euporie.core.app.cast

euporie.core.app.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.app.create_app_session

euporie.core.app.**create_app_session**(*input: Input | None = None*, *output: Output | None = None*) → Generator[AppSession, None, None]

Create a separate AppSession.

This is useful if there can be multiple individual `AppSession`s going on. Like in the case of an Telnet/SSH server.

### euporie.core.app.create_input

euporie.core.app.**create_input**(*stdin:* TextIO | *None = None*, *always_prefer_tty: bool = False*) → Input

Create the appropriate *Input* object for the current os/environment.

**Parameters**

**always_prefer_tty** – When set, if *sys.stdin* is connected to a Unix *pipe*, check whether *sys.stdout* or *sys.stderr* are connected to a pseudo terminal. If so, open the tty for reading instead of reading for *sys.stdin*. (We can open *stdout* or *stderr* for reading, this is how a *$PAGER* works.)

### euporie.core.app.create_output

euporie.core.app.**create_output**(*stdout:* TextIO | *None* = *None*, *always_prefer_tty:* *bool* = *False*) → Output

Return an `Output` instance for the command line.

**Parameters**

- **stdout** – The stdout object
- **always_prefer_tty** – When set, look for *sys.stderr* if *sys.stdout* is not a TTY. Useful if *sys.stdout* is redirected to a file, but we still want user input and output on the terminal.

  By default, this is *False*. If *sys.stdout* is not a terminal (maybe it's redirected to a file), then a *PlainTextOutput* will be returned. That way, tools like *print_formatted_text* will write plain text into that file.

### euporie.core.app.get_app

euporie.core.app.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.app.get_mime

euporie.core.app.**get_mime**(*path:* Path | *str*) → str | None

Attempt to determine the mime-type of a path.

### euporie.core.app.get_style_by_name

euporie.core.app.**get_style_by_name**(*name*)

Return a style class by its short name. The names of the builtin styles are listed in `pygments.styles.STYLE_MAP`.

Will raise *pygments.util.ClassNotFound* if no style of that name is found.

### euporie.core.app.load_cpr_bindings

euporie.core.app.**load_cpr_bindings**() → *KeyBindings*

### euporie.core.app.load_emacs_bindings

euporie.core.app.**load_emacs_bindings**() → *KeyBindingsBase*

Some e-macs extensions.

### euporie.core.app.load_emacs_search_bindings

euporie.core.app.**load_emacs_search_bindings**() → *KeyBindingsBase*

### euporie.core.app.load_emacs_shift_selection_bindings

euporie.core.app.**load_emacs_shift_selection_bindings**() → *KeyBindingsBase*

Bindings to select text with shift + cursor movements

### euporie.core.app.load_ptk_basic_bindings

euporie.core.app.**load_ptk_basic_bindings**() → *KeyBindings*

### euporie.core.app.load_ptk_mouse_bindings

euporie.core.app.**load_ptk_mouse_bindings**() → *KeyBindings*

Key bindings, required for mouse support. (Mouse events enter through the key binding system.)

### euporie.core.app.load_registered_bindings

euporie.core.app.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.core.app.load_vi_bindings

euporie.core.app.**load_vi_bindings**() → *KeyBindingsBase*

Vi extensions.

# Overview of Readline Vi commands: # http://www.catonmat.net/download/bash-vi-editing-mode-cheat-sheet.pdf

### euporie.core.app.load_vi_search_bindings

euporie.core.app.**load_vi_search_bindings**() → *KeyBindingsBase*

### euporie.core.app.merge_key_bindings

euporie.core.app.**merge_key_bindings**(*bindings: Sequence[*KeyBindingsBase*]*) → _MergedKeyBindings

Merge multiple `Keybinding` objects together.

Usage:

```
bindings = merge_key_bindings([bindings1, bindings2, ...])
```

### euporie.core.app.merge_style_transformations

euporie.core.app.**merge_style_transformations**(*style_transformations:* *Sequence[StyleTransformation]*) → StyleTransformation

Merge multiple transformations together.

### euporie.core.app.merge_styles

euporie.core.app.**merge_styles**(*styles:* *list[*prompt_toolkit.styles.base.BaseStyle*]*) → _MergedStyle

Merge multiple *Style* objects.

### euporie.core.app.parse_path

euporie.core.app.**parse_path**(*path:* *str* | *PathLike*, *resolve:* *bool* = *True*) → *Path*

Parse and resolve a path.

### euporie.core.app.register_bindings

euporie.core.app.**register_bindings**(*bindings:* *dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.app.set_app

euporie.core.app.**set_app**(*app:* Application*[Any]*) → Generator[None, None, None]

Context manager that sets the given `Application` active in an *AppSession*.

This should only be called by the *Application* itself. The application will automatically be active while its running. If you want the application to be active in other threads/coroutines, where that's not the case, use *contextvars.copy_context()*, or use *Application.context* to run it in the appropriate context.

### euporie.core.app.setup_logs

euporie.core.app.**setup_logs**(*config:* Config | *None* = *None*) → None

Configure the logger for euporie.

### euporie.core.app.style_from_pygments_cls

euporie.core.app.**style_from_pygments_cls**(*pygments_style_cls: type[PygmentsStyle]*) → *Style*

Shortcut to create a *Style* instance from a Pygments style class and a style dictionary.

Example:

```
from prompt_toolkit.styles.from_pygments import style_from_pygments_cls
from pygments.styles import get_style_by_name
style = style_from_pygments_cls(get_style_by_name('monokai'))
```

**Parameters**

**pygments_style_cls** – Pygments style class to start from.

### euporie.core.app.to_container

euporie.core.app.**to_container**(*container: AnyContainer*) → *Container*

Make sure that the given object is a *Container*.

### euporie.core.app.to_filter

euporie.core.app.**to_filter**(*bool_or_filter: Union[*Filter*, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### Classes

| | |
|---|---|
| *Application*([layout, style, ...]) | The main Application class! This glues everything together. |
| *BaseApp*([title, set_title, leave_graphics, ...]) | All euporie apps. |
| *BaseStyle*() | Abstract base class for prompt_toolkit styles. |
| *ChainedList*(*lists) | A list-like class which chains multiple lists. |
| *CliFormatter*(command[, languages]) | Format using an external command. |
| *ColorDepth*(value[, names, module, qualname, ...]) | Possible color depth values for the output. |
| *ColorPalette*() | Define a collection of colors. |
| *CompletionsMenu*([max_height, scroll_offset, ...]) | A custom completions menu. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalKeyBindings*(key_bindings[, filter]) | Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.::. |
| *ConditionalStyleTransformation*(...) | Apply the style transformation depending on a condition. |
| *Config*() | A configuration store. |
| *ConfiguredClipboard*(app) | Use a clipboard determined by euporie's configuration. |
| *CursorConfig*() | Determine which cursor mode to use. |
| *CursorShape*(value[, names, module, ...]) | |
| *CursorShapeConfig*() | |
| *DummyStyle*() | A style that doesn't style anything. |
| *Float*(content[, top, right, bottom, left, ...]) | Float for use in a *FloatContainer*. |
| *FloatContainer*(content, floats[, modal, ...]) | A *FloatContainer* which uses :py`BoundedWritePosition`s. |

continues on next page

Table 4 – continued from previous page

| | |
|---|---|
| *KeyProcessor*(*args, **kwargs) | A subclass of prompt_toolkit's keyprocessor. |
| *Layout*(container[, focused_element]) | The layout for a prompt_toolkit `Application`. |
| *LspClient*(name, command[, languages, settings]) | A client for communicating with LSP servers. |
| *MicroState*() | Mutable class to hold Micro specific state. |
| *Point*(x, y) | |
| *PtkVt100_Output* | alias of *Vt100_Output* |
| *PurePath*(*args, **kwargs) | Base class for manipulating paths without I/O. |
| *Renderer*(style, output[, full_screen, ...]) | Renderer with modifications. |
| *SetDefaultColorStyleTransformation*(fg, bg) | Set default foreground/background color for output that doesn't specify anything. |
| *Shadow*(body) | Draw a shadow underneath/behind this container. |
| *Style*(style_rules) | Create a `Style` instance from a list of style rules. |
| *SwapLightAndDarkStyleTransformation*() | Turn dark colors into light colors and the other way around. |
| *TerminalInfo*(input_, output, config) | A class to gather and hold information about the terminal. |
| *UPath*(*args[, protocol]) | |
| *ViState*() | Mutable class to hold the state of the Vi navigation. |
| *Vt100Parser*(*args, **kwargs) | A Vt100Parser which checks input against additional key patterns. |
| *Vt100_Output*(stdout, get_size[, term, ...]) | A Vt100 output which enables SGR pixel mouse positioning. |
| *WeakSet*([data]) | |
| *WeakValueDictionary*([other]) | Mapping class that references values weakly. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.app.Application

**class** euporie.core.app.**Application**(*layout:* Layout | *None* = *None*, *style:* BaseStyle | *None* = *None*, *include_default_pygments_style: FilterOrBool* = *True*, *style_transformation: StyleTransformation* | *None* = *None*, *key_bindings:* KeyBindingsBase | *None* = *None*, *clipboard:* Clipboard | *None* = *None*, *full_screen: bool* = *False*, *color_depth:* ColorDepth | *Callable[[],* ColorDepth | *None]* | *None* = *None*, *mouse_support: FilterOrBool* = *False*, *enable_page_navigation_bindings: None* | *FilterOrBool* = *None*, *paste_mode: FilterOrBool* = *False*, *editing_mode:* EditingMode = *EditingMode.EMACS*, *erase_when_done: bool* = *False*, *reverse_vi_search_direction: FilterOrBool* = *False*, *min_redraw_interval: float* | *int* | *None* = *None*, *max_render_postpone_time: float* | *int* | *None* = *0.01*, *refresh_interval: float* | *None* = *None*, *terminal_size_polling_interval: float* | *None* = *0.5*, *cursor: AnyCursorShapeConfig* = *None*, *on_reset: ApplicationEventHandler[_AppResult]* | *None* = *None*, *on_invalidate: ApplicationEventHandler[_AppResult]* | *None* = *None*, *before_render: ApplicationEventHandler[_AppResult]* | *None* = *None*, *after_render: ApplicationEventHandler[_AppResult]* | *None* = *None*, *input: Input* | *None* = *None*, *output: Output* | *None* = *None*)

The main Application class! This glues everything together.

**Parameters**

- **layout** – A `Layout` instance.
- **key_bindings** – `KeyBindingsBase` instance for the key bindings.
- **clipboard** – `Clipboard` to use.
- **full_screen** – When True, run the application on the alternate screen buffer.
- **color_depth** – Any *`ColorDepth`* value, a callable that returns a *`ColorDepth`* or *None* for default.
- **erase_when_done** – (bool) Clear the application output when it finishes.
- **reverse_vi_search_direction** – Normally, in Vi mode, a '/' searches forward and a '?' searches backward. In Readline mode, this is usually reversed.
- **min_redraw_interval** – Number of seconds to wait between redraws. Use this for applications where *invalidate* is called a lot. This could cause a lot of terminal output, which some terminals are not able to process.

  *None* means that every *invalidate* will be scheduled right away (which is usually fine).

  When one *invalidate* is called, but a scheduled redraw of a previous *invalidate* call has not been executed yet, nothing will happen in any case.
- **max_render_postpone_time** – When there is high CPU (a lot of other scheduled calls), postpone the rendering max x seconds. '0' means: don't postpone. '.5' means: try to draw at least twice a second.
- **refresh_interval** – Automatically invalidate the UI every so many seconds. When *None* (the default), only invalidate when *invalidate* has been called.
- **terminal_size_polling_interval** – Poll the terminal size every so many seconds. Useful if the applications runs in a thread other then then main thread where SIGWINCH can't be handled, or on Windows.

Filters:

**Parameters**

- **mouse_support** – (`Filter` or boolean). When True, enable mouse support.
- **paste_mode** – `Filter` or boolean.
- **editing_mode** – *`EditingMode`*.
- **enable_page_navigation_bindings** – When *True*, enable the page navigation key bindings. These include both Emacs and Vi bindings like page-up, page-down and so on to scroll through pages. Mostly useful for creating an editor or other full screen applications. Probably, you don't want this for the implementation of a REPL. By default, this is enabled if *full_screen* is set.

Callbacks (all of these should accept an `Application` object as input.)

**Parameters**

- **on_reset** – Called during reset.
- **on_invalidate** – Called when the UI has been invalidated.
- **before_render** – Called right before rendering.
- **after_render** – Called right after rendering.

I/O: (Note that the preferred way to change the input/output is by creating an *AppSession* with the required input/output objects. If you need multiple applications running at the same time, you have to create a separate *AppSession* using a *with create_app_session():* block.

**Parameters**

- **input** – `Input` instance.
- **output** – `Output` instance. (Probably Vt100_Output or Win32Output.)

Usage:

app = Application(…) app.run()

# Or await app.run_async()

### euporie.core.app.BaseApp

**class** euporie.core.app.**BaseApp**(*title: str | None = None*, *set_title: bool = True*, *leave_graphics: FilterOrBool = True*, *extend_renderer_height: FilterOrBool = False*, *extend_renderer_width: FilterOrBool = False*, *enable_page_navigation_bindings: FilterOrBool | None = True*, ***kwargs: Any*)

All euporie apps.

The base euporie application class.

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

### euporie.core.app.BaseStyle

**class** euporie.core.app.**BaseStyle**

Abstract base class for prompt_toolkit styles.

### euporie.core.app.ChainedList

**class** euporie.core.app.**ChainedList**(**lists: Iterable[T]*)

A list-like class which chains multiple lists.

### euporie.core.app.CliFormatter

**class** euporie.core.app.**CliFormatter**(*command: list[str]*, *languages: set[str] | None = None*)

Format using an external command.

### euporie.core.app.ColorDepth

**class** euporie.core.app.**ColorDepth**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Possible color depth values for the output.

### euporie.core.app.ColorPalette

**class** euporie.core.app.**ColorPalette**

Define a collection of colors.

### euporie.core.app.CompletionsMenu

**class** euporie.core.app.**CompletionsMenu**(*max_height: int | None = 16*, *scroll_offset: int | Callable[[], int] = 1*, *extra_filter: FilterOrBool = True*, *z_index: int = 100000000*)

A custom completions menu.

### euporie.core.app.Condition

**class** euporie.core.app.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.app.ConditionalKeyBindings

**class** euporie.core.app.**ConditionalKeyBindings**(*key_bindings: KeyBindingsBase*, *filter: Union[Filter, bool] = True*)

Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.:

```
@Condition
def setting_is_true():
    return True  # or False

registry = ConditionalKeyBindings(key_bindings, setting_is_true)
```

When new key bindings are added to this object. They are also enable/disabled according to the given *filter*.

**Parameters**

- **registries** – List of *KeyBindings* objects.
- **filter** – Filter object.

### euporie.core.app.ConditionalStyleTransformation

**class** euporie.core.app.**ConditionalStyleTransformation**(*style_transformation: StyleTransformation*, *filter: Union[*Filter, *bool]*)

Apply the style transformation depending on a condition.

### euporie.core.app.Config

**class** euporie.core.app.**Config**

A configuration store.

### euporie.core.app.ConfiguredClipboard

**class** euporie.core.app.**ConfiguredClipboard**(*app:* BaseApp)

Use a clipboard determined by euporie's configuration.

### euporie.core.app.CursorConfig

**class** euporie.core.app.**CursorConfig**

Determine which cursor mode to use.

### euporie.core.app.CursorShape

**class** euporie.core.app.**CursorShape**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.app.CursorShapeConfig

**class** euporie.core.app.**CursorShapeConfig**

### euporie.core.app.DummyStyle

**class** euporie.core.app.**DummyStyle**

A style that doesn't style anything.

### euporie.core.app.Float

**class** euporie.core.app.**Float**(*content: AnyContainer*, *top: int | None = None*, *right: int | None = None*, *bottom: int | None = None*, *left: int | None = None*, *width: int | Callable[[], int] | None = None*, *height: int | Callable[[], int] | None = None*, *xcursor: bool = False*, *ycursor: bool = False*, *attach_to_window: AnyContainer | None = None*, *hide_when_covering_content: bool = False*, *allow_cover_cursor: bool = False*, *z_index: int = 1*, *transparent: bool = False*)

Float for use in a *FloatContainer*. Except for the *content* parameter, all other options are optional.

**Parameters**

- **content** – *Container* instance.
- **width** – *Dimension* or callable which returns a *Dimension*.
- **height** – *Dimension* or callable which returns a *Dimension*.
- **left** – Distance to the left edge of the *FloatContainer*.
- **right** – Distance to the right edge of the *FloatContainer*.
- **top** – Distance to the top of the *FloatContainer*.
- **bottom** – Distance to the bottom of the *FloatContainer*.
- **attach_to_window** – Attach to the cursor from this window, instead of the current window.
- **hide_when_covering_content** – Hide the float when it covers content underneath.
- **allow_cover_cursor** – When *False*, make sure to display the float below the cursor. Not on top of the indicated position.
- **z_index** – Z-index position. For a Float, this needs to be at least one. It is relative to the z_index of the parent container.
- **transparent** – *Filter* indicating whether this float needs to be drawn transparently.

### euporie.core.app.FloatContainer

**class** euporie.core.app.**FloatContainer**(*content: AnyContainer*, *floats: list[*Float*]*, *modal: bool = False*, *key_bindings:* KeyBindingsBase *| None = None*, *style: str | Callable[[], str] = ''*, *z_index: int | None = None*)

A *FloatContainer* which uses :py`BoundedWritePosition`s.

### euporie.core.app.KeyProcessor

**class** euporie.core.app.**KeyProcessor**(**args: Any*, ***kwargs: Any*)

A subclass of prompt_toolkit's keyprocessor.

This adds an exception to the auto-flush timeout so that the input is flushed immediately if the key pressed is the escape key.

### euporie.core.app.Layout

**class** euporie.core.app.**Layout**(*container: AnyContainer*, *focused_element: FocusableElement | None = None*)

The layout for a prompt_toolkit `Application`. This also keeps track of which user control is focused.

**Parameters**

- **container** – The "root" container for the layout.
- **focused_element** – element to be focused initially. (Can be anything the *focus* function accepts.)

### euporie.core.app.LspClient

**class** euporie.core.app.**LspClient**(*name: str*, *command: str*, *languages: Sequence[str] | None = None*, *settings: dict | None = None*)

A client for communicating with LSP servers.

### euporie.core.app.MicroState

**class** euporie.core.app.**MicroState**

Mutable class to hold Micro specific state.

### euporie.core.app.Point

**class** euporie.core.app.**Point**(*x*, *y*)

### euporie.core.app.PtkVt100_Output

euporie.core.app.**PtkVt100_Output**

alias of *Vt100_Output*

### euporie.core.app.PurePath

**class** euporie.core.app.**PurePath**(**args*, ***kwargs*)

Base class for manipulating paths without I/O.

PurePath represents a filesystem path and offers operations which don't imply any actual filesystem I/O. Depending on your system, instantiating a PurePath will return either a PurePosixPath or a PureWindowsPath object. You can also instantiate either of these classes directly, regardless of your system.

### euporie.core.app.Renderer

**class** euporie.core.app.**Renderer**(*style: BaseStyle*, *output: Output*, *full_screen: bool = False*, *mouse_support: FilterOrBool = False*, *cpr_not_supported_callback: Callable[[], None] | None = None*, *extend_height: FilterOrBool = False*, *extend_width: FilterOrBool = False*)

Renderer with modifications.

### euporie.core.app.SetDefaultColorStyleTransformation

**class** euporie.core.app.**SetDefaultColorStyleTransformation**(*fg: Union[str, Callable[[], str]]*, *bg: Union[str, Callable[[], str]]*)

Set default foreground/background color for output that doesn't specify anything. This is useful for overriding the terminal default colors.

**Parameters**

- **fg** – Color string or callable that returns a color string for the foreground.
- **bg** – Like *fg*, but for the background.

### euporie.core.app.Shadow

**class** euporie.core.app.**Shadow**(*body: AnyContainer*)

Draw a shadow underneath/behind this container.

This is a globally configurable version of the `prompt_toolkit.widows.base.Shadow` class.

### euporie.core.app.Style

**class** euporie.core.app.**Style**(*style_rules: list[tuple[str, str]]*)

Create a `Style` instance from a list of style rules.

The *style_rules* is supposed to be a list of ('classnames', 'style') tuples. The classnames are a whitespace separated string of class names and the style string is just like a Pygments style definition, but with a few additions: it supports 'reverse' and 'blink'.

Later rules always override previous rules.

Usage:

```
Style([
    ('title', '#ff0000 bold underline'),
    ('something-else', 'reverse'),
    ('class1 class2', 'reverse'),
])
```

The `from_dict` classmethod is similar, but takes a dictionary as input.

### euporie.core.app.SwapLightAndDarkStyleTransformation

**class** euporie.core.app.**SwapLightAndDarkStyleTransformation**

Turn dark colors into light colors and the other way around.

This is meant to make color schemes that work on a dark background usable on a light background (and the other way around).

Notice that this doesn't swap foreground and background like "reverse" does. It turns light green into dark green and the other way around. Foreground and background colors are considered individually.

Also notice that when <reverse> is used somewhere and no colors are given in particular (like what is the default for the bottom toolbar), then this doesn't change anything. This is what makes sense, because when the 'default' color is chosen, it's what works best for the terminal, and reverse works good with that.

### euporie.core.app.TerminalInfo

**class** euporie.core.app.**TerminalInfo**(*input_: Input*, *output: Output*, *config:* Config)

A class to gather and hold information about the terminal.

### euporie.core.app.UPath

**class** euporie.core.app.**UPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

### euporie.core.app.ViState

**class** euporie.core.app.**ViState**

Mutable class to hold the state of the Vi navigation.

### euporie.core.app.Vt100Parser

**class** euporie.core.app.**Vt100Parser**(**args: Any*, ***kwargs: Any*)

A Vt100Parser which checks input against additional key patterns.

### euporie.core.app.Vt100_Output

**class** euporie.core.app.**Vt100_Output**(*stdout:* TextIO, *get_size: Callable[[],* Size*]*, *term: str | None = None*, *default_color_depth:* prompt_toolkit.output.color_depth.ColorDepth *| None = None*, *enable_bell: bool = True*, *enable_cpr: bool = True*)

A Vt100 output which enables SGR pixel mouse positioning.

### euporie.core.app.WeakSet

**class** euporie.core.app.**WeakSet**(*data=None*)

### euporie.core.app.WeakValueDictionary

**class** euporie.core.app.**WeakValueDictionary**(*other=()*, */*, ***kw*)

Mapping class that references values weakly.

Entries in the dictionary will be discarded when no strong reference to the value exists anymore

### euporie.core.app.Window

**class** euporie.core.app.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.core.app.partial

**class** euporie.core.app.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.app.**BaseApp**(*title: str | None = None*, *set_title: bool = True*, *leave_graphics: FilterOrBool = True*, *extend_renderer_height: FilterOrBool = False*, *extend_renderer_width: FilterOrBool = False*, *enable_page_navigation_bindings: FilterOrBool | None = True*, ***kwargs: Any*)

Bases: *Application*

All euporie apps.

The base euporie application class.

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

**async cancel_and_wait_for_background_tasks**() → None

Cancel all background tasks, and wait for the cancellation to complete. 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.)

**cleanup**(*signum: int*, *frame: FrameType | None*) → None

Restore the state of the terminal on unexpected exit.

**cleanup_closed_tab**(*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:* Tab | *None* = *None*) → None

Close a notebook tab.

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

**property 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.

**color_palette: *ColorPalette***

**cpr_not_supported_callback**() → None

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

**create_background_task**(*coroutine: Coroutine[Any, Any, None]*) → Task[None]

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

Given that we still support Python versions before 3.11, we can't use task groups (and exception groups), because of that, these background tasks are not allowed to raise exceptions. If they do, we'll call the default exception handler from the event loop.

If at some point, we have Python 3.11 as the minimum supported Python version, then we can use a *TaskGroup* (with the lifetime of *Application.run_async()*, and run run the background tasks in there.

This is not threadsafe.

**create_merged_style**() → *BaseStyle*

Generate a new merged style for the application.

Using a dynamic style has serious performance issues, so instead we update the style on the renderer directly when it changes in *self.update_style*

**Returns**
Return a combined style to use for the application

**property current_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: SearchState**

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

**draw**(*render_as_done: bool = True*) → None

Draw the app without focus, leaving the cursor below the drawn output.

**exit**(*result: Optional[_AppResult] = None*, *exception: BaseException | type[BaseException] | None = 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.)

**focus_tab**(*tab: Tab*) → None

Make a tab visible and focuses it.

**get_edit_mode**() → *EditingMode*

Return the editing mode enum defined in the configuration.

**get_file_tab**(*path: Path*) → type[*Tab*] | None

Return the tab to use for a file path.

**get_file_tabs**(*path: Path*) → list[type[*Tab*]]

Return the tab to use for a file path.

**get_language_lsps**(*language: str*) → list[*euporie.core.lsp.LspClient*]

Return the approprrate LSP clients for a given language.

**get_used_style_strings**() → list[str]

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

**async classmethod interact**(*ssh_session: PromptToolkitSSHSession*) → None

Run the app asynchronously for the hub SSH server.

**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

Launch the app.

**load_container**() → *FloatContainer*

Load the root container for this application.

**Returns**

The root container for this app

**classmethod load_input**() → Input

Create the input for this application to use.

Ensures the TUI app always tries to run in a TTY.

**Returns**

A prompt-toolkit input instance

**load_key_bindings**() → None

Load the application's key bindings.

**classmethod load_output**() → Output

Create the output for this application to use.

Ensures the TUI app always tries to run in a TTY.

**Returns**

A prompt-toolkit output instance

**log_stdout_level: str = 'CRITICAL'**

**mouse_position: *Point***

**name: str**

**open_file**(*path:* Path, *read_only: bool = False*, *tab_class: type[*Tab*] | None = None*) → None

Create 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
- **tab_class** – The tab type to use to open the file

**open_files**() → None

Open the files defined in the configuration.

**pause_rendering**() → None

Block rendering, but allows input to be processed.

The first line prevents the display being drawn, and the second line means the key processor continues to process keys. We need this as we need to wait for the results of terminal queries which come in as key events.

This is used to prevent flicker when we update the styles based on terminal feedback.

**post_load**() → None

Allow subclasses to define additional loading steps.

**pre_run**(*app:* prompt_toolkit.application.application.Application | *None* = *None*) → None

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

**print_text**(*text: AnyFormattedText*, *style:* BaseStyle | *None* = *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.

**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.

**resume_rendering**() → None

Reume rendering the app.

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

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

This will run the application in a fresh asyncio event loop.

**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_appp().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: Callable[[], None] | None = None*, *set_exception_handler: bool = True*, *handle_sigint: bool = True*, *slow_callback_duration: float = 0.5*) → _AppResult

Run the application.

**async run_system_command**(*command: str*, *wait_for_enter: bool = True*, *display_before_text: AnyFormattedText = ''*, *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.

**shutdown_lsps**() → None

Shut down all the remaining LSP servers.

**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 syntax_theme: str**

Calculate the current syntax theme.

**property tab: Tab | None**

Return the currently selected tab container object.

**property tab_idx: int**

Get 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 uses 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.

**property title: str**

The application's title.

**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_edit_mode**(*setting: Setting | None = None*) → None

Set the keybindings for editing mode.

**update_style**(*query: TerminalQuery | Setting | None = None*) → None

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

**vi_state**
Vi state. (For Vi key bindings.)

**class** euporie.core.app.**CursorConfig**
Bases: *CursorShapeConfig*

Determine which cursor mode to use.

**get_cursor_shape**(*app:* Application*[Any]*) → *CursorShape*
Return the cursor shape to be used in the current state.

### euporie.core.border

Define border styles.

### Functions

| | |
|---|---|
| *NamedTuple*(typename[, fields]) | Typed version of namedtuple. |
| *get_grid_char*(key) | Return the character represented by a combination of `LineStyles`. |
| *lru_cache*([maxsize, typed]) | Least-recently-used cache decorator. |
| *total_ordering*(cls) | Class decorator that fills in missing ordering methods |

### euporie.core.border.NamedTuple

euporie.core.border.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)
Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

### euporie.core.border.get_grid_char

euporie.core.border.**get_grid_char**(*key:* GridChar) → str

Return the character represented by a combination of `LineStyles`.

### euporie.core.border.lru_cache

euporie.core.border.**lru_cache**(*maxsize=128*, *typed=False*)

Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

### euporie.core.border.total_ordering

euporie.core.border.**total_ordering**(*cls*)

Class decorator that fills in missing ordering methods

### Classes

| | |
|---|---|
| *DiLineStyle*([top, right, bottom, left]) | A description of a cell border: a *LineStyle* for each edge. |
| *DirectionFlags*([north, east, south, west]) | Flag which indicate the connection of a grid node. |
| *Enum*(value[, names, module, qualname, type, ...]) | Create a collection of name/value pairs. |
| *FastDictCache*(get_value[, size]) | Fast, lightweight cache which keeps at most *size* items. |
| *GridChar*(north, east, south, west) | Repreentation of a grid node character. |
| *GridPart*(value[, names, module, qualname, ...]) | Define the component characters of a grid. |
| *GridStyle*(line_style, mask) | A collection of characters which can be used to draw a grid. |
| *LineStyle*(name, rank[, parent, visible]) | Define a line style which can be used to draw grids. |
| *Mask*(mask) | A mask which allows selection of a subset of a grid. |
| *Masks*() | A collection of default masks. |

### euporie.core.border.DiLineStyle

**class** euporie.core.border.**DiLineStyle**(*top:* LineStyle = *LineStyle(None)*, *right:* LineStyle = *LineStyle(None)*, *bottom:* LineStyle = *LineStyle(None)*, *left:* LineStyle = *LineStyle(None)*)

A description of a cell border: a *LineStyle* for each edge.

### euporie.core.border.DirectionFlags

**class** euporie.core.border.**DirectionFlags**(*north: bool = False*, *east: bool = False*, *south: bool = False*, *west: bool = False*)

Flag which indicate the connection of a grid node.

### euporie.core.border.Enum

**class** euporie.core.border.**Enum**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes – see the documentation for details.

### euporie.core.border.FastDictCache

**class** euporie.core.border.**FastDictCache**(*get_value: Callable[[...], _V]*, *size: int = 1000000*)

Fast, lightweight cache which keeps at most *size* items. It will discard the oldest items in the cache first.

The cache is a dictionary, which doesn't keep track of access counts. It is perfect to cache little immutable objects which are not expensive to create, but where a dictionary lookup is still much faster than an object instantiation.

**Parameters**

**get_value** – Callable that's called in case of a missing key.

### euporie.core.border.GridChar

**class** euporie.core.border.**GridChar**(*north:* LineStyle, *east:* LineStyle, *south:* LineStyle, *west:* LineStyle)

Repreentation of a grid node character.

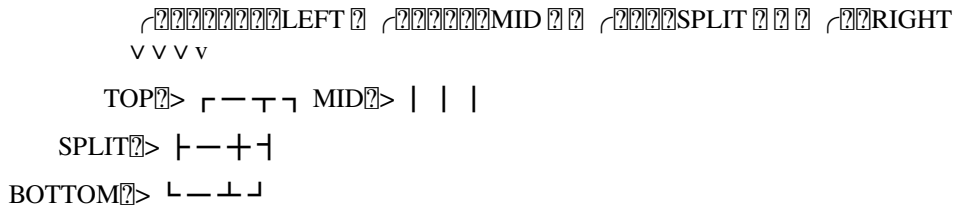
The four compass points represent the line style joining from the given direction.

### euporie.core.border.GridPart

**class** euporie.core.border.**GridPart**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Define the component characters of a grid.

Character naming works as follows:




### euporie.core.border.GridStyle

**class** euporie.core.border.**GridStyle**(*line_style: ~euporie.core.border.LineStyle = LineStyle(None)*, *mask: ~euporie.core.border.Mask = <euporie.core.border.Mask object>*)

A collection of characters which can be used to draw a grid.

### euporie.core.border.LineStyle

**class** euporie.core.border.**LineStyle**(*name: str*, *rank: tuple[int, int]*, *parent:* euporie.core.border.LineStyle | *None = None*, *visible: bool = True*)

Define a line style which can be used to draw grids.

`GridStyle`s can be created from a :class:`LineStyle` by accessing an attribute with the name of a default mask from *Masks*.

### euporie.core.border.Mask

**class** euporie.core.border.**Mask**(*mask: dict[*euporie.core.border.GridPart, euporie.core.border.DirectionFlags*]*)

A mask which allows selection of a subset of a grid.

Masks can be combined to construct more complex masks.

### euporie.core.border.Masks

**class** euporie.core.border.**Masks**

A collection of default masks.

**class** euporie.core.border.**DiLineStyle**(*top:* LineStyle = *LineStyle(None)*, *right:* LineStyle = *LineStyle(None)*, *bottom:* LineStyle = *LineStyle(None)*, *left:* LineStyle = *LineStyle(None)*)

Bases: NamedTuple

A description of a cell border: a *LineStyle* for each edge.

**bottom: *LineStyle***

Alias for field number 2

**count**(*value*, */*)

Return number of occurrences of value.

**classmethod from_value**(*value:* LineStyle) → *DiLineStyle*

Construct an instance from a single value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**left: *LineStyle***

Alias for field number 3

**right: *LineStyle***

Alias for field number 1

**top: *LineStyle***

Alias for field number 0

**class** euporie.core.border.**DirectionFlags**(*north: bool = False*, *east: bool = False*, *south: bool = False*, *west: bool = False*)

Bases: NamedTuple

Flag which indicate the connection of a grid node.

**count**(*value*, */*)

Return number of occurrences of value.

**east: bool**

Alias for field number 1

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**north: bool**

Alias for field number 0

**south: bool**

Alias for field number 2

**west: bool**

Alias for field number 3

**class** euporie.core.border.**GridChar**(*north:* LineStyle, *east:* LineStyle, *south:* LineStyle, *west:* LineStyle)

Bases: NamedTuple

Repreentation of a grid node character.

The four compass points represent the line style joining from the given direction.

**count**(*value*, */*)

Return number of occurrences of value.

**east: *LineStyle***

Alias for field number 1

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**north: *LineStyle***

Alias for field number 0

**south: *LineStyle***

Alias for field number 2

**west: *LineStyle***

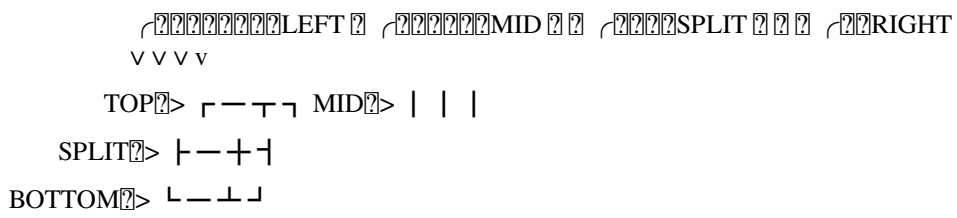
Alias for field number 3

**class** euporie.core.border.**GridPart**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *Enum*

Define the component characters of a grid.

Character naming works as follows:




**BOTTOM_LEFT = 12**

**BOTTOM_MID = 13**

**BOTTOM_RIGHT = 15**

**BOTTOM_SPLIT = 14**

**MID_LEFT = 4**

**MID_MID = 5**

**MID_RIGHT = 7**

**MID_SPLIT = 6**

**SPLIT_LEFT = 8**

**SPLIT_MID = 9**

**SPLIT_RIGHT = 11**

**SPLIT_SPLIT = 10**

**TOP_LEFT = 0**

**TOP_MID = 1**

**TOP_RIGHT = 3**

**TOP_SPLIT = 2**

**class** euporie.core.border.**GridStyle**(*line_style: ~euporie.core.border.LineStyle = LineStyle(None)*, *mask: ~euporie.core.border.Mask = <euporie.core.border.Mask object>*)

Bases: object

A collection of characters which can be used to draw a grid.

**property BOTTOM: _BorderLineChars**

Allow dotted attribute access to the bottom grid row.

**property HORIZONTAL: str**

For compatibility with prompt_toolkit.widgets.base.Border.

**property MID: _BorderLineChars**

Allow dotted attribute access to the mid grid row.

**property SPLIT: _BorderLineChars**

Allow dotted attribute access to the split grid row.

**property TOP: _BorderLineChars**

Allow dotted attribute access to the top grid row.

**property VERTICAL: str**

For compatibility with prompt_toolkit.widgets.base.Border.

**class** euporie.core.border.**LineStyle**(*name: str*, *rank: tuple[int, int]*, *parent:* euporie.core.border.LineStyle | *None = None*, *visible: bool = True*)

Bases: object

Define a line style which can be used to draw grids.

GridStyle`s can be created from a :class:`LineStyle by accessing an attribute with the name of a default mask from *Masks*.

**class** euporie.core.border.**Mask**(*mask: dict[*euporie.core.border.GridPart, euporie.core.border.DirectionFlags*]*)

Bases: object

A mask which allows selection of a subset of a grid.

Masks can be combined to construct more complex masks.

**class** euporie.core.border.**Masks**

Bases: object

A collection of default masks.

**bottom_edge = <euporie.core.border.Mask object>**

**center_edge = <euporie.core.border.Mask object>**

**corners = <euporie.core.border.Mask object>**

**grid = <euporie.core.border.Mask object>**

**inner = <euporie.core.border.Mask object>**

**left_edge = <euporie.core.border.Mask object>**

**middle_edge = <euporie.core.border.Mask object>**

**outer = <euporie.core.border.Mask object>**

**right_edge = <euporie.core.border.Mask object>**

**top_edge = <euporie.core.border.Mask object>**

euporie.core.border.**get_grid_char**(*key:* GridChar) → str

Return the character represented by a combination of LineStyles.

### euporie.core.clipboard

Module concerning clipboard access and manipulation.

### Functions

| | |
|---|---|
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *determine_clipboard*() | Determine the OS/platform and set the copy() and paste() functions accordingly. |

### euporie.core.clipboard.add_setting

euporie.core.clipboard.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.clipboard.determine_clipboard

euporie.core.clipboard.**determine_clipboard**()

Determine the OS/platform and set the copy() and paste() functions accordingly.

### Classes

| | |
|---|---|
| *Clipboard*() | Abstract baseclass for clipboards. |
| *ClipboardData*([text, type]) | Text on the clipboard. |
| *ConfiguredClipboard*(app) | Use a clipboard determined by euporie's configuration. |
| *InMemoryClipboard*([data, max_size]) | Default clipboard implementation. |
| *Osc52Clipboard*(app) | Clipboard that syncs with the system clipboard using OSC52 escape codes. |
| *PyperclipClipboard*() | Clipboard that synchronizes with the Windows/Mac/Linux system clipboard, using the pyperclip module. |
| *SelectionType*(value[, names, module, ...]) | Type of selection. |
| *Vt100_Output*(stdout, get_size[, term, ...]) | A Vt100 output which enables SGR pixel mouse positioning. |

### euporie.core.clipboard.Clipboard

**class** euporie.core.clipboard.**Clipboard**

Abstract baseclass for clipboards. (An implementation can be in memory, it can share the X11 or Windows keyboard, or can be persistent.)

### euporie.core.clipboard.ClipboardData

**class** euporie.core.clipboard.**ClipboardData**(*text: str = ''*, *type: SelectionType = SelectionType.CHARACTERS*)

Text on the clipboard.

**Parameters**

- **text** – string
- **type** – *SelectionType*

### euporie.core.clipboard.ConfiguredClipboard

**class** euporie.core.clipboard.**ConfiguredClipboard**(*app:* BaseApp)

Use a clipboard determined by euporie's configuration.

### euporie.core.clipboard.InMemoryClipboard

**class** euporie.core.clipboard.**InMemoryClipboard**(*data:* prompt_toolkit.clipboard.base.ClipboardData | *None* = *None*, *max_size: int* = *60*)

Default clipboard implementation. Just keep the data in memory.

This implements a kill-ring, for Emacs mode.

### euporie.core.clipboard.Osc52Clipboard

**class** euporie.core.clipboard.**Osc52Clipboard**(*app:* BaseApp)

Clipboard that syncs with the system clipboard using OSC52 escape codes.

### euporie.core.clipboard.PyperclipClipboard

**class** euporie.core.clipboard.**PyperclipClipboard**

Clipboard that synchronizes with the Windows/Mac/Linux system clipboard, using the pyperclip module.

### euporie.core.clipboard.SelectionType

**class** euporie.core.clipboard.**SelectionType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Type of selection.

### euporie.core.clipboard.Vt100_Output

**class** euporie.core.clipboard.**Vt100_Output**(*stdout:* TextIO, *get_size: Callable[[],* Size*]*, *term: str* | *None* = *None*, *default_color_depth:* prompt_toolkit.output.color_depth.ColorDepth | *None* = *None*, *enable_bell: bool* = *True*, *enable_cpr: bool* = *True*)

A Vt100 output which enables SGR pixel mouse positioning.

**class** euporie.core.clipboard.**ConfiguredClipboard**(*app:* BaseApp)

Bases: *Clipboard*

Use a clipboard determined by euporie's configuration.

**get_clipboard**(*setting:* Setting) → None

Determine which clipboard to use.

**get_data**() → *ClipboardData*

Return clipboard data.

**rotate**() → None

For Emacs mode, rotate the kill ring.

**set_data**(*data:* ClipboardData) → None

Set data to the clipboard.

**set_text**(*text: str*) → None

Shortcut for setting plain text on clipboard.

**class** euporie.core.clipboard.**Osc52Clipboard**(*app:* BaseApp)

Bases: *Clipboard*

Clipboard that syncs with the system clipboard using OSC52 escape codes.

**get_data**() → *ClipboardData*

Retrieve clipboard data.

**rotate**() → None

For Emacs mode, rotate the kill ring.

**set_data**(*data:* ClipboardData) → None

Set clipboard data.

**set_text**(*text: str*) → None

Shortcut for setting plain text on clipboard.

### euporie.core.comm

Sub-module for handling kernel Comm messages.

### Modules

| | |
|---|---|
| *euporie.core.comm.base* | Define the base class for a Comm object and it's representation. |
| *euporie.core.comm.ipywidgets* | Define representations for ipywidget comms. |
| *euporie.core.comm.registry* | Contain a registry of Comm target classes. |

### euporie.core.comm.base

Define the base class for a Comm object and it's representation.

## Functions

| | |
|---|---|
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |
| *get_app*() | Get the current active (running) Application. |

### euporie.core.comm.base.abstractmethod

euporie.core.comm.base.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):
…

### euporie.core.comm.base.get_app

euporie.core.comm.base.**get_app**() → *BaseApp*

Get the current active (running) Application.

## Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *Comm*(comm_container, comm_id, data, buffers) | A base-class for all comm objects, which support syncing traits with the kernel. |
| *CommView*(container[, setters]) | Hold a container and methods to update its attributes. |
| *Datum*(data, *args, **kwargs) | Class for storing and converting display data. |
| *Display*(datum[, height, width, focusable, ...]) | Rich output displays. |
| *UnimplementedComm*(comm_container, comm_id, ...) | Represent a Comm object which is not implemented in euporie.core. |
| *WeakKeyDictionary*([dict]) | Mapping class that references keys weakly. |

### euporie.core.comm.base.ABCMeta

**class** euporie.core.comm.base.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.comm.base.Comm

**class** euporie.core.comm.base.**Comm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A base-class for all comm objects, which support syncing traits with the kernel.

### euporie.core.comm.base.CommView

**class** euporie.core.comm.base.**CommView**(*container: AnyContainer*, *setters:* Mapping*[str, Callable[..., None]]* | *None* = *None*)

Hold a container and methods to update its attributes.

### euporie.core.comm.base.Datum

**class** euporie.core.comm.base.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Class for storing and converting display data.

### euporie.core.comm.base.Display

**class** euporie.core.comm.base.**Display**(*datum:* Datum, *height: AnyDimension = None*, *width: AnyDimension = None*, *focusable: FilterOrBool = False*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *always_hide_cursor: FilterOrBool = True*, *scrollbar: FilterOrBool = True*, *scrollbar_autohide: FilterOrBool = True*, *dont_extend_height: FilterOrBool = True*, *dont_extend_width: FilterOrBool = False*, *style: str* | *Callable[[], str] = ''*)

Rich output displays.

A container for displaying rich output data.

### euporie.core.comm.base.UnimplementedComm

**class** euporie.core.comm.base.**UnimplementedComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Represent a Comm object which is not implemented in euporie.core.

### euporie.core.comm.base.WeakKeyDictionary

**class** euporie.core.comm.base.**WeakKeyDictionary**(*dict=None*)

Mapping class that references keys weakly.

Entries in the dictionary will be discarded when there is no longer a strong reference to the key. This can be used to associate additional data with an object owned by other parts of an application without adding attributes to those objects. This can be especially useful with objects that override attribute accesses.

**class** euporie.core.comm.base.**Comm**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: object

A base-class for all comm objects, which support syncing traits with the kernel.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new *CommView* for this Comm.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**abstract process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Process a comm_msg data / buffers.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.base.**CommView**(*container: AnyContainer*, *setters: Mapping[str, Callable[..., None]] | None = None*)

Bases: object

Hold a container and methods to update its attributes.

**update**(*changes: dict[str, Any]*) → None

Update the view to reflect changes in the Comm.

Calls any setter functions defined for the changed keys with the changed values as arguments.

**Parameters**

**changes** – A dictionary mapping changed key names to new values.

**class** euporie.core.comm.base.**UnimplementedComm**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *Comm*

Represent a Comm object which is not implemented in euporie.core.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new *CommView* for this Comm.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Doe nothing when data is received.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

### euporie.core.comm.ipywidgets

Define representations for `ipywidget` comms.

### Functions

| | |
|---|---|
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |
| *copy*(x) | Shallow copy operation on arbitrary Python objects. |
| *open_comm_ipywidgets*(comm_container, ...) | Create a new Comm for an `ipywidgets` widget. |
| *standard_b64encode*(s) | Encode bytes-like object s using the standard Base64 alphabet. |

### euporie.core.comm.ipywidgets.abstractmethod

euporie.core.comm.ipywidgets.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):

…

### euporie.core.comm.ipywidgets.copy

euporie.core.comm.ipywidgets.**copy**(*x*)

Shallow copy operation on arbitrary Python objects.

See the module's __doc__ string for more info.

### euporie.core.comm.ipywidgets.open_comm_ipywidgets

euporie.core.comm.ipywidgets.**open_comm_ipywidgets**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*) → *IpyWidgetComm*

Create a new Comm for an `ipywidgets` widget.

The relevant widget model is selected based on the model name given in the `comm_open` message data.

**Parameters**

- **comm_container** – The notebook this Comm belongs to
- **comm_id** – The ID of the Comm
- **data** – The data field from the `comm_open` message

- **buffers** – The buffers field from the `comm_open` message

**Returns**
The initialized widget Comm object.

### euporie.core.comm.ipywidgets.standard_b64encode

euporie.core.comm.ipywidgets.**standard_b64encode**(*s*)

Encode bytes-like object s using the standard Base64 alphabet.

The result is returned as a bytes object.

### Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *AccordionModel*(comm_container, comm_id, ...) | An accoridon layout ipywidget. |
| *AccordionSplit*(children, titles[, active, ...]) | A container which switches between children using expandable sections. |
| *BeforeInput*(text[, style]) | Insert text before the input. |
| *BoundedFloatTextModel*(comm_container, ...) | An float textbox ipwidget with upper and lower bounds. |
| *BoundedIntTextModel*(comm_container, comm_id, ...) | An integer textbox ipwidget with upper and lower bounds. |
| *Box*(body[, padding, padding_left, ...]) | Add padding around a container. |
| *BoxModel*(comm_container, comm_id, data, buffers) | A box layout ipywidget (basically the same a HBox). |
| *Button*(text, on_click, None] \| None = None, ...) | A clickable button widget. |
| *ButtonModel*(comm_container, comm_id, data, ...) | A Button ipywidget. |
| *CellOutputArea*(json, parent[, style]) | An area below a cell where one or more cell outputs can be shown. |
| *Checkbox*([text, on_click, prefix, style, ...]) | A toggleable checkbox widget. |
| *CheckboxModel*(comm_container, comm_id, data, ...) | A checkbox ipywidget. |
| *ColorPickerModel*(comm_container, comm_id, ...) | A color picker ipywidget. |
| *ComboboxModel*(comm_container, comm_id, data, ...) | A combobox input widget. |
| *Comm*(comm_container, comm_id, data, buffers) | A base-class for all comm objects, which support syncing traits with the kernel. |
| *CommView*(container[, setters]) | Hold a container and methods to update its attributes. |
| *Condition*(func) | Turn any callable into a Filter. |
| *DatePickerModel*(comm_container, comm_id, ...) | A date-picker ipywidget. |
| *Datum*(data, *args, **kwargs) | Class for storing and converting display data. |
| *Decimal*([value, context]) | Construct a new Decimal object. |
| *DiBool*([top, right, bottom, left]) | A tuple of four bools with directions. |
| *Display*(datum[, height, width, focusable, ...]) | Rich output displays. |
| *Dropdown*(options[, labels, index, indices, ...]) | A dropdown widget, allowing selection of an item from a menu of options. |
| *DropdownModel*(comm_container, comm_id, data, ...) | An ipywidget allowing an item to be selected using a drop-down menu. |
| *FloatLogOptionsMixin*() | A mixin for ipywidgets which accept a value from range of exponents. |

continues on next page

Table 5 – continued from previous page

| | |
|---|---|
| *FloatLogSliderModel*(comm_container, comm_id, ...) | A slider ipywidget that accepts a single value on a log scale. |
| *FloatOptionsMixin*() | A mixin for ipywidgets which accept a range of float values. |
| *FloatProgressModel*(comm_container, comm_id, ...) | A progress bar ipywidget that accepts float values. |
| *FloatRangeSliderModel*(comm_container, ...) | A slider ipywidget that accepts a range of float values. |
| *FloatSliderModel*(comm_container, comm_id, ...) | A slider ipywidget that accepts a single float value. |
| *FloatTextModel*(comm_container, comm_id, ...) | A float textbox ipwidget. |
| *FocusedStyle*(body[, style_focus, style_hover]) | Apply a style to child containers when focused or hovered. |
| *HBoxModel*(comm_container, comm_id, data, buffers) | A horizontal layout ipywidget. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *HTMLMathModel*(comm_container, comm_id, data, ...) | Alia for *HTMLModel*, which can render maths. |
| *HTMLModel*(comm_container, comm_id, data, buffers) | A label ipywidget which displays HTML. |
| *ImageModel*(comm_container, comm_id, data, ...) | A ipywidget which displays an image. |
| *IntOptionsMixin*() | A mixin for ipywidgets which accept a range of integer values. |
| *IntProgressModel*(comm_container, comm_id, ...) | A progress bar ipywidget that accepts integer values. |
| *IntRangeSliderModel*(comm_container, comm_id, ...) | A slider ipywidget that accepts a range of integer values. |
| *IntSliderModel*(comm_container, comm_id, ...) | A slider ipywidget that accepts a single integer value. |
| *IntTextModel*(comm_container, comm_id, data, ...) | An integer textbox ipwidget. |
| *IpyWidgetComm*(comm_container, comm_id, data, ...) | A Comm object which represents ipython widgets. |
| *Label*(value[, style, html]) | A label widget which displays rich text. |
| *LabelModel*(comm_container, comm_id, data, ...) | A label ipwidget. |
| *LabelledWidget*(body, label[, style, ...]) | A widget which applies a label to another widget. |
| *LayoutIpyWidgetComm*(comm_container, comm_id, ...) | Base class for layout widgets with children. |
| *MsgCallbacks* | Typed dictionary for named message callbacks. |
| *NumberTextBoxIpyWidgetComm*(comm_container, ...) | Base class for text-box ipywidgets with numerical values. |
| *OutputModel*(comm_container, comm_id, data, ...) | An Output ipywidget. |
| *Progress*([start, stop, step, value, ...]) | A progress-bar widget. |
| *ProgressIpyWidgetComm*(comm_container, ...) | The base class for progress bar ipywidgets. |
| *RadioButtonsModel*(comm_container, comm_id, ...) | An ipywidget allowing an item to be selected using radio buttons. |
| *RangeSliderIpyWidgetComm*(comm_container, ...) | Base class for range slider ipywidgets. |
| *ReferencedSplit*(split, children, *args, **kwargs) | A split container which maintains a reference to it's children. |
| *Select*(options, labels, index, indices, ...) | A select widget, which allows one or more items to be selected from a list. |
| *SelectModel*(comm_container, comm_id, data, ...) | An ipywidget allowing a value to be selected from a list of options. |
| *SelectMultipleModel*(comm_container, comm_id, ...) | An ipywidget allowing one or more value to be selected from a list of options. |
| *SelectableIpyWidgetComm*(comm_container, ...) | Base class for selectable ipywidgets. |
| *SelectionRangeSliderModel*(comm_container, ...) | A slider ipywidget where one or more of a list of options can be selected. |

continues on next page

Table 5 – continued from previous page

| | |
|---|---|
| *SelectionSliderModel*(comm_container, ...) | A slider ipywidget where one of a list of options can be selected. |
| *Slider*(options, labels, index, indices, ...) | A slider widget with an optional editable readout. |
| *SliderIpyWidgetComm*(comm_container, comm_id, ...) | Base class for slider ipywidgets. |
| *Swatch*(color, str] =, width, height, style, ...) | An widget which displays a given color. |
| *TabModel*(comm_container, comm_id, data, buffers) | A tabbed layout ipywidget. |
| *TabbedSplit*(children, titles, active, style, ...) | A container which switches between children using tabs. |
| *Text*([text, style, height, min_height, ...]) | A text input widget. |
| *TextBoxIpyWidgetComm*(comm_container, ...) | A mixin for ipywidgets which use text-box entry. |
| *TextModel*(comm_container, comm_id, data, buffers) | A text input widget. |
| *TextareaModel*(comm_container, comm_id, data, ...) | A text input widget. |
| *ToggleButton*(text, on_click, ...) | A toggleable button widget. |
| *ToggleButtonModel*(comm_container, comm_id, ...) | A toggleable button ipywidget. |
| *ToggleButtons*(options, labels, index, ...) | A widget where an option is selected using mutually exclusive toggle-buttons. |
| *ToggleButtonsModel*(comm_container, comm_id, ...) | An ipywidget where a single value can be selected using togglable buttons. |
| *ToggleableIpyWidgetComm*(comm_container, ...) | Base class for toggleable ipywidgets. |
| *UnimplementedModel*(comm_container, comm_id, ...) | An ipywidget used to represent unimplemented widgets. |
| *VBoxModel*(comm_container, comm_id, data, buffers) | A vertical layout ipywidget. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *ValidModel*(comm_container, comm_id, data, ...) | A validity indicator ipywidget. |
| *date* | date(year, month, day) --> date object |
| *datetime*(year, month, day[, hour[, minute[, ...) | The year, month and day arguments are required. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.comm.ipywidgets.ABCMeta

**class** euporie.core.comm.ipywidgets.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.comm.ipywidgets.AccordionModel

**class** euporie.core.comm.ipywidgets.**AccordionModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An accoridon layout ipywidget.

### euporie.core.comm.ipywidgets.AccordionSplit

**class** euporie.core.comm.ipywidgets.**AccordionSplit**(*children: Sequence[AnyContainer]*, *titles: Sequence[AnyFormattedText]*, *active: int = 0*, *style: str | Callable[[], str] = 'class:tab-split'*, *on_change: Callable[[StackedSplit], None] | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*)

A container which switches between children using expandable sections.

### euporie.core.comm.ipywidgets.BeforeInput

**class** euporie.core.comm.ipywidgets.**BeforeInput**(*text: AnyFormattedText*, *style: str = ''*)

Insert text before the input.

**Parameters**

- **text** – This can be either plain text or formatted text (or a callable that returns any of those).
- **style** – style to be applied to this prompt/prefix.

### euporie.core.comm.ipywidgets.BoundedFloatTextModel

**class** euporie.core.comm.ipywidgets.**BoundedFloatTextModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An float textbox ipwidget with upper and lower bounds.

### euporie.core.comm.ipywidgets.BoundedIntTextModel

**class** euporie.core.comm.ipywidgets.**BoundedIntTextModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An integer textbox ipwidget with upper and lower bounds.

### euporie.core.comm.ipywidgets.Box

**class** euporie.core.comm.ipywidgets.**Box**(*body: AnyContainer*, *padding: AnyDimension = None*, *padding_left: AnyDimension = None*, *padding_right: AnyDimension = None*, *padding_top: AnyDimension = None*, *padding_bottom: AnyDimension = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *char: None | str | Callable[[], str] = None*, *modal: bool = False*, *key_bindings:* KeyBindings | *None = None*)

Add padding around a container.

This also makes sure that the parent can provide more space than required by the child. This is very useful when wrapping a small element with a fixed size into a `VSplit` or `HSplit` object. The `HSplit` and `VSplit` try to make sure to adapt respectively the width and height, possibly shrinking other elements. Wrapping something in a `Box` makes it flexible.

**Parameters**

- **body** – Another container object.
- **padding** – The margin to be used around the body. This can be
- **padding_left** (*overridden by*) – *padding_bottom*.
- **padding_right`** – *padding_bottom*.
- **and** (*padding_top*) – *padding_bottom*.
- **style** – A style string.
- **char** – Character to be used for filling the space around the body. (This is supposed to be a character with a terminal width of 1.)

### euporie.core.comm.ipywidgets.BoxModel

**class** euporie.core.comm.ipywidgets.**BoxModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A box layout ipywidget (basically the same a HBox).

### euporie.core.comm.ipywidgets.Button

**class** euporie.core.comm.ipywidgets.**Button**(*text: AnyFormattedText*, *on_click: Callable[[Button], None] | None = None*, *on_mouse_down: Callable[[Button], None] | None = None*, *disabled: FilterOrBool = False*, *width: int | None = None*, *style: str | Callable[[], str] = 'class:input'*, *border: GridStyle | None = □□ □ □□ □□□□□ □□*, *show_borders: DiBool | None = None*, *selected: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *mouse_handler: Callable[[MouseEvent], NotImplementedOrNone] | None = None*)

A clickable button widget.

### euporie.core.comm.ipywidgets.ButtonModel

**class** euporie.core.comm.ipywidgets.**ButtonModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A Button ipywidget.

### euporie.core.comm.ipywidgets.CellOutputArea

**class** euporie.core.comm.ipywidgets.**CellOutputArea**(*json: list[dict[str, Any]]*, *parent: OutputParent | None*, *style: str = ''*)

An area below a cell where one or more cell outputs can be shown.

### euporie.core.comm.ipywidgets.Checkbox

**class** euporie.core.comm.ipywidgets.**Checkbox**(*text: AnyFormattedText = ''*, *on_click: Callable[[ToggleableWidget], None] | None = None*, *prefix: tuple[str, str] = ('', '')*, *style: str = 'class:input'*, *selected: bool = False*, *disabled: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*)

A toggleable checkbox widget.

### euporie.core.comm.ipywidgets.CheckboxModel

**class** euporie.core.comm.ipywidgets.**CheckboxModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A checkbox ipywidget.

### euporie.core.comm.ipywidgets.ColorPickerModel

**class** euporie.core.comm.ipywidgets.**ColorPickerModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A color picker ipywidget.

### euporie.core.comm.ipywidgets.ComboboxModel

**class** euporie.core.comm.ipywidgets.**ComboboxModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A combobox input widget.

### euporie.core.comm.ipywidgets.Comm

**class** euporie.core.comm.ipywidgets.**Comm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A base-class for all comm objects, which support syncing traits with the kernel.

### euporie.core.comm.ipywidgets.CommView

**class** euporie.core.comm.ipywidgets.**CommView**(*container: AnyContainer*, *setters:* Mapping*[str, Callable[..., None]]* | *None* = *None*)

Hold a container and methods to update its attributes.

### euporie.core.comm.ipywidgets.Condition

**class** euporie.core.comm.ipywidgets.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.comm.ipywidgets.DatePickerModel

**class** euporie.core.comm.ipywidgets.**DatePickerModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A date-picker ipywidget.

### euporie.core.comm.ipywidgets.Datum

**class** euporie.core.comm.ipywidgets.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Class for storing and converting display data.

### euporie.core.comm.ipywidgets.Decimal

**class** euporie.core.comm.ipywidgets.**Decimal**(*value='0'*, *context=None*)

Construct a new Decimal object. 'value' can be an integer, string, tuple, or another Decimal object. If no value is given, return Decimal('0'). The context does not affect the conversion and is only passed to determine if the InvalidOperation trap is active.

### euporie.core.comm.ipywidgets.DiBool

**class** euporie.core.comm.ipywidgets.**DiBool**(*top: bool = False*, *right: bool = False*, *bottom: bool = False*, *left: bool = False*)

A tuple of four bools with directions.

### euporie.core.comm.ipywidgets.Display

**class** euporie.core.comm.ipywidgets.**Display**(*datum: Datum*, *height: AnyDimension = None*, *width: AnyDimension = None*, *focusable: FilterOrBool = False*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *always_hide_cursor: FilterOrBool = True*, *scrollbar: FilterOrBool = True*, *scrollbar_autohide: FilterOrBool = True*, *dont_extend_height: FilterOrBool = True*, *dont_extend_width: FilterOrBool = False*, *style: str | Callable[[], str] = ''*)

Rich output displays.

A container for displaying rich output data.

### euporie.core.comm.ipywidgets.Dropdown

**class** euporie.core.comm.ipywidgets.**Dropdown**(*options: list[Any]*, *labels: Sequence[AnyFormattedText] | None = None*, *index: int | None = None*, *indices: list[int] | None = None*, *n_values: int | None = None*, *multiple: FilterOrBool = False*, *max_count: int | None = None*, *on_change: Callable[[SelectableWidget], None] | None = None*, *style: str | Callable[[], str] = 'class:input'*, *arrow: str = '▼'*, *disabled: FilterOrBool = False*)

A dropdown widget, allowing selection of an item from a menu of options.

### euporie.core.comm.ipywidgets.DropdownModel

**class** euporie.core.comm.ipywidgets.**DropdownModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An ipywidget allowing an item to be selected using a drop-down menu.

### euporie.core.comm.ipywidgets.FloatLogOptionsMixin

**class** euporie.core.comm.ipywidgets.**FloatLogOptionsMixin**

A mixin for ipywidgets which accept a value from range of exponents.

### euporie.core.comm.ipywidgets.FloatLogSliderModel

**class** euporie.core.comm.ipywidgets.**FloatLogSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A slider ipywidget that accepts a single value on a log scale.

### euporie.core.comm.ipywidgets.FloatOptionsMixin

**class** euporie.core.comm.ipywidgets.**FloatOptionsMixin**

A mixin for ipywidgets which accept a range of float values.

### euporie.core.comm.ipywidgets.FloatProgressModel

**class** euporie.core.comm.ipywidgets.**FloatProgressModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A progress bar ipywidget that accepts float values.

### euporie.core.comm.ipywidgets.FloatRangeSliderModel

**class** euporie.core.comm.ipywidgets.**FloatRangeSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A slider ipywidget that accepts a range of float values.

### euporie.core.comm.ipywidgets.FloatSliderModel

**class** euporie.core.comm.ipywidgets.**FloatSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A slider ipywidget that accepts a single float value.

### euporie.core.comm.ipywidgets.FloatTextModel

**class** euporie.core.comm.ipywidgets.**FloatTextModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A float textbox ipwidget.

### euporie.core.comm.ipywidgets.FocusedStyle

**class** euporie.core.comm.ipywidgets.**FocusedStyle**(*body: AnyContainer*, *style_focus: str | Callable[[], str] = 'class:focused'*, *style_hover: str | Callable[[], str] = ''*)

Apply a style to child containers when focused or hovered.

### euporie.core.comm.ipywidgets.HBoxModel

**class** euporie.core.comm.ipywidgets.**HBoxModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A horizontal layout ipywidget.

### euporie.core.comm.ipywidgets.HSplit

**class** euporie.core.comm.ipywidgets.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

### euporie.core.comm.ipywidgets.HTMLMathModel

**class** euporie.core.comm.ipywidgets.**HTMLMathModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Alia for *HTMLModel*, which can render maths.

### euporie.core.comm.ipywidgets.HTMLModel

**class** euporie.core.comm.ipywidgets.**HTMLModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A label ipywidget which displays HTML.

### euporie.core.comm.ipywidgets.ImageModel

**class** euporie.core.comm.ipywidgets.**ImageModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A ipywidget which displays an image.

### euporie.core.comm.ipywidgets.IntOptionsMixin

**class** euporie.core.comm.ipywidgets.**IntOptionsMixin**

A mixin for ipywidgets which accept a range of integer values.

### euporie.core.comm.ipywidgets.IntProgressModel

**class** euporie.core.comm.ipywidgets.**IntProgressModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A progress bar ipywidget that accepts integer values.

### euporie.core.comm.ipywidgets.IntRangeSliderModel

**class** euporie.core.comm.ipywidgets.**IntRangeSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A slider ipywidget that accepts a range of integer values.

### euporie.core.comm.ipywidgets.IntSliderModel

**class** euporie.core.comm.ipywidgets.**IntSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A slider ipywidget that accepts a single integer value.

### euporie.core.comm.ipywidgets.IntTextModel

**class** euporie.core.comm.ipywidgets.**IntTextModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An integer textbox ipwidget.

### euporie.core.comm.ipywidgets.IpyWidgetComm

**class** euporie.core.comm.ipywidgets.**IpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A Comm object which represents ipython widgets.

### euporie.core.comm.ipywidgets.Label

**class** euporie.core.comm.ipywidgets.**Label**(*value: AnyFormattedText*, *style: str | Callable[[], str] = 'class:input'*, *html: FilterOrBool = False*)

A label widget which displays rich text.

### euporie.core.comm.ipywidgets.LabelModel

**class** euporie.core.comm.ipywidgets.**LabelModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A label ipwidget.

### euporie.core.comm.ipywidgets.LabelledWidget

**class** euporie.core.comm.ipywidgets.**LabelledWidget**(*body: AnyContainer*, *label: AnyFormattedText*, *style: str = 'class:input'*, *vertical: FilterOrBool = False*, *html: FilterOrBool = False*)

A widget which applies a label to another widget.

### euporie.core.comm.ipywidgets.LayoutIpyWidgetComm

**class** euporie.core.comm.ipywidgets.**LayoutIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Base class for layout widgets with children.

### euporie.core.comm.ipywidgets.MsgCallbacks

**class** euporie.core.comm.ipywidgets.**MsgCallbacks**

Typed dictionary for named message callbacks.

### euporie.core.comm.ipywidgets.NumberTextBoxIpyWidgetComm

**class** euporie.core.comm.ipywidgets.**NumberTextBoxIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Base class for text-box ipywidgets with numerical values.

### euporie.core.comm.ipywidgets.OutputModel

**class** euporie.core.comm.ipywidgets.**OutputModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An Output ipywidget.

### euporie.core.comm.ipywidgets.Progress

**class** euporie.core.comm.ipywidgets.**Progress**(*start: float | int = 0*, *stop: float | int = 100*, *step: float | int = 1*, *value: float | int = 0*, *vertical: FilterOrBool = False*, *style: str | Callable[[], str] = 'class:input'*)

A progress-bar widget.

### euporie.core.comm.ipywidgets.ProgressIpyWidgetComm

**class** euporie.core.comm.ipywidgets.**ProgressIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

The base class for progress bar ipywidgets.

### euporie.core.comm.ipywidgets.RadioButtonsModel

**class** euporie.core.comm.ipywidgets.**RadioButtonsModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An ipywidget allowing an item to be selected using radio buttons.

### euporie.core.comm.ipywidgets.RangeSliderIpyWidgetComm

**class** euporie.core.comm.ipywidgets.**RangeSliderIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Base class for range slider ipywidgets.

### euporie.core.comm.ipywidgets.ReferencedSplit

**class** euporie.core.comm.ipywidgets.**ReferencedSplit**(*split: type[_Split]*, *children: Sequence[AnyContainer]*, **args: Any*, ***kwargs: Any*)

A split container which maintains a reference to it's children.

### euporie.core.comm.ipywidgets.Select

**class** euporie.core.comm.ipywidgets.**Select**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input,select', rows: int | None = 3, prefix: tuple[str, str] = ('', ''), border: GridStyle | None = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None, disabled: FilterOrBool = False, dont_extend_width: FilterOrBool = True, dont_extend_height: FilterOrBool = True*)

A select widget, which allows one or more items to be selected from a list.

### euporie.core.comm.ipywidgets.SelectModel

**class** euporie.core.comm.ipywidgets.**SelectModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An ipywidget allowing a value to be selected from a list of options.

### euporie.core.comm.ipywidgets.SelectMultipleModel

**class** euporie.core.comm.ipywidgets.**SelectMultipleModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An ipywidget allowing one or more value to be selected from a list of options.

### euporie.core.comm.ipywidgets.SelectableIpyWidgetComm

**class** euporie.core.comm.ipywidgets.**SelectableIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Base class for selectable ipywidgets.

### euporie.core.comm.ipywidgets.SelectionRangeSliderModel

**class** euporie.core.comm.ipywidgets.**SelectionRangeSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A slider ipywidget where one or more of a list of options can be selected.

### euporie.core.comm.ipywidgets.SelectionSliderModel

**class** euporie.core.comm.ipywidgets.**SelectionSliderModel**(*comm_container: KernelTab, comm_id: str, data: dict, buffers: Sequence[bytes]*)

A slider ipywidget where one of a list of options can be selected.

### euporie.core.comm.ipywidgets.Slider

**class** euporie.core.comm.ipywidgets.**Slider**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None, vertical: FilterOrBool = False, show_arrows: FilterOrBool = True, arrows: tuple[AnyFormattedText, AnyFormattedText] = ('-', '+'), show_readout: FilterOrBool = True, disabled: FilterOrBool = False*)

A slider widget with an optional editable readout.

### euporie.core.comm.ipywidgets.SliderIpyWidgetComm

**class** euporie.core.comm.ipywidgets.**SliderIpyWidgetComm**(*comm_container: KernelTab, comm_id: str, data: dict, buffers: Sequence[bytes]*)

Base class for slider ipywidgets.

### euporie.core.comm.ipywidgets.Swatch

**class** euporie.core.comm.ipywidgets.**Swatch**(*color: str | Callable[[], str] = '#FFFFFF', width: int = 2, height: int = 1, style: str = 'class:swatch', border: GridStyle = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None*)

An widget which displays a given color.

### euporie.core.comm.ipywidgets.TabModel

**class** euporie.core.comm.ipywidgets.**TabModel**(*comm_container: KernelTab, comm_id: str, data: dict, buffers: Sequence[bytes]*)

A tabbed layout ipywidget.

### euporie.core.comm.ipywidgets.TabbedSplit

**class** euporie.core.comm.ipywidgets.**TabbedSplit**(*children: Sequence[AnyContainer], titles: Sequence[AnyFormattedText], active: int = 0, style: str | Callable[[], str] = 'class:tab-split', on_change: Callable[[StackedSplit], None] | None = None, width: AnyDimension = None, height: AnyDimension = None, border: GridStyle = □□□□□ □ □□ □□□□ □□□□□, show_borders: DiBool | None = None*)

A container which switches between children using tabs.

### euporie.core.comm.ipywidgets.Text

**class** euporie.core.comm.ipywidgets.**Text**(*text: str = '', style: str = 'class:input', height: int = 1, min_height: int = 1, multiline: FilterOrBool = False, expand: FilterOrBool = True, width: int | None = None, completer: Completer | None = None, options: list[str] | Callable[[], list[str]] | None = None, show_borders: DiBool | None = None, on_text_changed: Callable[[Buffer], None] | None = None, validation: Callable[[str], bool] | None = None, accept_handler: BufferAcceptHandler | None = None, placeholder: str | None = None, lexer: Lexer | None = None, input_processors: Sequence[Processor] | None = None, disabled: FilterOrBool = False, password: FilterOrBool = False, wrap_lines: FilterOrBool = False, prompt: AnyFormattedText | None = None*)

A text input widget.

### euporie.core.comm.ipywidgets.TextBoxIpyWidgetComm

**class** euporie.core.comm.ipywidgets.**TextBoxIpyWidgetComm**(*comm_container: KernelTab, comm_id: str, data: dict, buffers: Sequence[bytes]*)

A mixin for ipywidgets which use text-box entry.

### euporie.core.comm.ipywidgets.TextModel

**class** euporie.core.comm.ipywidgets.**TextModel**(*comm_container: KernelTab, comm_id: str, data: dict, buffers: Sequence[bytes]*)

A text input widget.

### euporie.core.comm.ipywidgets.TextareaModel

**class** euporie.core.comm.ipywidgets.**TextareaModel**(*comm_container: KernelTab, comm_id: str, data: dict, buffers: Sequence[bytes]*)

A text input widget.

### euporie.core.comm.ipywidgets.ToggleButton

**class** euporie.core.comm.ipywidgets.**ToggleButton**(*text: AnyFormattedText, on_click: Callable[[ToggleButton], None] | None = None, width: int | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None, selected: bool = False, disabled: FilterOrBool = False, key_bindings: KeyBindingsBase | None = None*)

A toggleable button widget.

### euporie.core.comm.ipywidgets.ToggleButtonModel

**class** euporie.core.comm.ipywidgets.**ToggleButtonModel**(*comm_container: KernelTab, comm_id: str, data: dict, buffers: Sequence[bytes]*)

A toggleable button ipywidget.

### euporie.core.comm.ipywidgets.ToggleButtons

**class** euporie.core.comm.ipywidgets.**ToggleButtons**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, disabled: FilterOrBool = False, vertical: FilterOrBool = False*)

A widget where an option is selected using mutually exclusive toggle-buttons.

### euporie.core.comm.ipywidgets.ToggleButtonsModel

**class** euporie.core.comm.ipywidgets.**ToggleButtonsModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An ipywidget where a single value can be selected using togglable buttons.

### euporie.core.comm.ipywidgets.ToggleableIpyWidgetComm

**class** euporie.core.comm.ipywidgets.**ToggleableIpyWidgetComm**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Base class for toggleable ipywidgets.

### euporie.core.comm.ipywidgets.UnimplementedModel

**class** euporie.core.comm.ipywidgets.**UnimplementedModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

An ipywidget used to represent unimplemented widgets.

### euporie.core.comm.ipywidgets.VBoxModel

**class** euporie.core.comm.ipywidgets.**VBoxModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A vertical layout ipywidget.

### euporie.core.comm.ipywidgets.VSplit

**class** euporie.core.comm.ipywidgets.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.comm.ipywidgets.ValidModel

**class** euporie.core.comm.ipywidgets.**ValidModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

A validity indicator ipywidget.

### euporie.core.comm.ipywidgets.date

**class** euporie.core.comm.ipywidgets.**date**

date(year, month, day) –> date object

### euporie.core.comm.ipywidgets.datetime

**class** euporie.core.comm.ipywidgets.**datetime**(*year*, *month*, *day*[, *hour*[, *minute*[, *second*[, *microsecond*[, *tzinfo*]]]]])

The year, month and day arguments are required. tzinfo may be None, or an instance of a tzinfo subclass. The remaining arguments may be ints.

### euporie.core.comm.ipywidgets.partial

**class** euporie.core.comm.ipywidgets.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.comm.ipywidgets.**AccordionModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *LayoutIpyWidgetComm*

An accoridon layout ipywidget.

**box_style**() → str

Convert the ipywidget box_style to a prompt_toolkit style string.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the accordion ipywidget.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**render_children**(*models: list[str]*, *parent: OutputParent*) → list[AnyContainer]

Create views for the child Comms in the layout.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_index**(*container:* StackedSplit) → None

Send a `comm_message` updating the selected index when it changes.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**BoundedFloatTextModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *FloatOptionsMixin*, *NumberTextBoxIpyWidgetComm*

An float textbox ipwidget with upper and lower bounds.

**create_view**(*parent: OutputParent*) → *CommView*
Create a new view of the numerical text-box ipywidget.

**data: dict[str, Any]**

**decr**(*button:* Button) → None
Decrement the widget's value by one step.

**default_rows = 1**

**incr**(*button:* Button) → None
Increment the widget's value by one step.

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*
Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → float | None
Enure the selected value is within the permitted range and is a float.

**property options: list[float]**
Generate a list of available options in a range of floats.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None
Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None
Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**validation**(*x: Any*) → bool
Enure the entered text can be normalized.

**value**() → str
Return the ipywidget's value.

**class** euporie.core.comm.ipywidgets.**BoundedIntTextModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IntOptionsMixin*, *NumberTextBoxIpyWidgetComm*

An integer textbox ipwidget with upper and lower bounds.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the numerical text-box ipywidget.

**data: dict[str, Any]**

**decr**(*button:* Button) → None

Decrement the widget's value by one step.

**default_rows = 1**

**incr**(*button:* Button) → None

Increment the widget's value by one step.

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → int | None

Enure the selected value is within the permitted range and is a integer.

**property options: list[int]**

Generate a list of available options in a range of integers.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None

Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**validation**(*x: Any*) → bool

Enure the entered text can be normalized.

**value**() → str

Return the ipywidget's value.

**class** euporie.core.comm.ipywidgets.**BoxModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *LayoutIpyWidgetComm*

A box layout ipywidget (basically the same a HBox).

**Split**

alias of *VSplit*

**box_style**() → str

Convert the ipywidget box_style to a prompt_toolkit style string.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the layout ipywidget.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**padding = 1**

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**render_children**(*models: list[str]*, *parent: OutputParent*) → list[AnyContainer]

Create views for the child Comms in the layout.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**ButtonModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

A Button ipywidget.

**button_style**() → str

Convert the ipywidget button_style to a prompt_toolkit style string.

**click**(*button: Button*) → None

Send a `comm_msg` describing a click event.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the button ipywidget.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**text**() → str

Generate the button text, optionally including an icon if specified.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**CheckboxModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *ToggleableIpyWidgetComm*

A checkbox ipywidget.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the checkbox ipywidget.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → float | None

Cat the container's selected value to a bool if possible.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**value_changed**(*button:* ToggleableWidget) → None

Send a `comm_message` updating the value when it changes.

**class** euporie.core.comm.ipywidgets.**ColorPickerModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *TextBoxIpyWidgetComm*

A color picker ipywidget.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the color-picker widget.

**default_rows = 1**

**format_color**() → str

Format a color as a hex code for display.

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: str*) → str | None

Return the color string if it is recognised as an allowed color.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None

Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**validation**(*x: Any*) → bool

Enure the entered text can be normalized.

**value**() → str

Return the ipywidget's value.

**class** euporie.core.comm.ipywidgets.**ComboboxModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *TextBoxIpyWidgetComm*

A combobox input widget.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the text-box ipywidget.

**default_rows = 1**

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: str*) → str | None

Enure that the entered text matches a permitted option, if required.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None

Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**validation**(*x: Any*) → bool

Enure the entered text can be normalized.

**value**() → str

Return the ipywidget's value.

**class** euporie.core.comm.ipywidgets.**DatePickerModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *TextBoxIpyWidgetComm*

A date-picker ipywidget.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the date-picker widget.

**default_rows = 1**

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: str*) → dict[str, int] | None

Attempt to convert entered text to the internal date representation.

**parse_date**(*value: dict[str, int]*) → *date*

Convert the internal date representation to a python date.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None

Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**validation**(*x: Any*) → bool

Enure the entered text can be normalized.

**value**() → str

Return the text to display in the widget's text area.

**class** euporie.core.comm.ipywidgets.**DropdownModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *SelectableIpyWidgetComm*

An ipywidget allowing an item to be selected using a drop-down menu.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the drop-down widget.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_index**(*container:* SelectableWidget) → None

Send a `comm_message` updating the selected index when it changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**FloatLogOptionsMixin**

Bases: *FloatOptionsMixin*

A mixin for ipywidgets which accept a value from range of exponents.

**data: dict[str, Any]**

**normalize**(*x: Any*) → float | None

Enure the selected value is within the permitted range and is a float.

**property options: list[float]**

Generate a list of available options in a range of log values.

**class** euporie.core.comm.ipywidgets.**FloatLogSliderModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *FloatLogOptionsMixin*, *SliderIpyWidgetComm*

A slider ipywidget that accepts a single value on a log scale.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the slider ipywidget.

**data: dict[str, Any]**

**property indices: list[Any]**

Return the selected index as a list.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → float | None

Enure the selected value is within the permitted range and is a float.

**property options: list[float]**

Generate a list of available options in a range of log values.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**set_value**(*slider: Slider*, *value: Any*) → None

Set the selected index when the ipywidget's selected value changes.

**target_name = 'jupyter.widget'**

**update_value**(*container: SelectableWidget*) → None

Send a `comm_message` updating the value when it changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**FloatOptionsMixin**

Bases: `object`

A mixin for ipywidgets which accept a range of float values.

**data: dict[str, Any]**

**normalize**(*x: Any*) → float | None

Enure the selected value is within the permitted range and is a float.

**property options: list[float]**

Generate a list of available options in a range of floats.

**class** euporie.core.comm.ipywidgets.**FloatProgressModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *FloatOptionsMixin*, *ProgressIpyWidgetComm*

A progress bar ipywidget that accepts float values.

**bar_style**() → str

Convert the ipywidget `bar_style` to a prompt_toolkit style string.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the progress bar.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → float | None

Enure the selected value is within the permitted range and is a float.

**property options: list[float]**

Generate a list of available options in a range of floats.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**FloatRangeSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *FloatOptionsMixin*, *RangeSliderIpyWidgetComm*

A slider ipywidget that accepts a range of float values.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the slider ipywidget.

**data: dict[str, Any]**

**property indices: list[int]**

Return the first and last selected indices.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → float | None

Enure the selected value is within the permitted range and is a float.

**property options: list[float]**

Generate a list of available options in a range of floats.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None
Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**set_value**(*slider:* Slider, *values: Any*) → None
Any float value is permitted - we might need to add an option.

**target_name = 'jupyter.widget'**

**update_value**(*slider:* SelectableWidget) → None
Send a `comm_message` updating the values when they change.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**FloatSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *FloatOptionsMixin*, *SliderIpyWidgetComm*

A slider ipywidget that accepts a single float value.

**create_view**(*parent: OutputParent*) → *CommView*
Create a new view of the slider ipywidget.

**data: dict[str, Any]**

**property indices: list[Any]**
Return the selected index as a list.

**new_view**(*parent: OutputParent*) → *CommView*
Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → float | None
Enure the selected value is within the permitted range and is a float.

**property options: list[float]**
Generate a list of available options in a range of floats.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None
Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**set_value**(*slider:* Slider, *value: Any*) → None
Set the selected index when the ipywidget's selected value changes.

**target_name = 'jupyter.widget'**

**update_value**(*container:* SelectableWidget) → None
Send a `comm_message` updating the value when it changes.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**FloatTextModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *FloatOptionsMixin*, *NumberTextBoxIpyWidgetComm*

A float textbox ipwidget.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the numerical text-box ipywidget.

**data: dict[str, Any]**

**decr**(*button:* Button) → None

Decrement the widget's value by one step.

**default_rows = 1**

**incr**(*button:* Button) → None

Increment the widget's value by one step.

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → float | None

Enure the selected value is within the permitted range and is a float.

**property options: list[float]**

Generate a list of available options in a range of floats.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None

Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**validation**(*x: Any*) → bool

Enure the entered text can be normalized.

**value**() → str

Return the ipywidget's value.

**class** euporie.core.comm.ipywidgets.**HBoxModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *BoxModel*

A horizontal layout ipywidget.

**Split**

alias of *VSplit*

**box_style**() → str

Convert the ipywidget box_style to a prompt_toolkit style string.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the layout ipywidget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**padding = 1**

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**render_children**(*models: list[str]*, *parent: OutputParent*) → list[AnyContainer]

Create views for the child Comms in the layout.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**HTMLMathModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *HTMLModel*

Alia for *HTMLModel*, which can render maths.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the HTML widget.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**HTMLModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

A label ipywidget which displays HTML.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the HTML widget.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**ImageModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

A ipywidget which displays an image.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the image widget.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**IntOptionsMixin**

Bases: `object`

A mixin for ipywidgets which accept a range of integer values.

**data: dict[str, Any]**

**normalize**(*x: Any*) → int | None

Enure the selected value is within the permitted range and is a integer.

**property options: list[int]**

Generate a list of available options in a range of integers.

**class** euporie.core.comm.ipywidgets.**IntProgressModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IntOptionsMixin*, *ProgressIpyWidgetComm*

A progress bar ipywidget that accepts integer values.

**bar_style**() → str

Convert the ipywidget `bar_style` to a prompt_toolkit style string.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the progress bar.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → int | None

Enure the selected value is within the permitted range and is a integer.

**property options: list[int]**

Generate a list of available options in a range of integers.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**IntRangeSliderModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IntOptionsMixin*, *RangeSliderIpyWidgetComm*

A slider ipywidget that accepts a range of integer values.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the slider ipywidget.

**data: dict[str, Any]**

**property indices: list[int]**

Return the first and last selected indices.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → int | None

Enure the selected value is within the permitted range and is a integer.

**property options: list[int]**

Generate a list of available options in a range of integers.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**set_value**(*slider:* Slider, *values: Any*) → None

Any float value is permitted - we might need to add an option.

**target_name = 'jupyter.widget'**

**update_value**(*slider:* SelectableWidget) → None

Send a `comm_message` updating the values when they change.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**IntSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IntOptionsMixin*, *SliderIpyWidgetComm*

A slider ipywidget that accepts a single integer value.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the slider ipywidget.

**data: dict[str, Any]**

**property indices: list[Any]**

Return the selected index as a list.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → int | None

Enure the selected value is within the permitted range and is a integer.

**property options: list[int]**

Generate a list of available options in a range of integers.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**set_value**(*slider:* Slider, *value: Any*) → None

Set the selected index when the ipywidget's selected value changes.

**target_name = 'jupyter.widget'**

**update_value**(*container:* SelectableWidget) → None

Send a `comm_message` updating the value when it changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**IntTextModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IntOptionsMixin*, *NumberTextBoxIpyWidgetComm*

An integer textbox ipwidget.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the numerical text-box ipywidget.

**data: dict[str, Any]**

**decr**(*button:* Button) → None

Decrement the widget's value by one step.

**default_rows = 1**

**incr**(*button:* Button) → None

Increment the widget's value by one step.

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → int | None

Enure the selected value is within the permitted range and is a integer.

**property options: list[int]**

Generate a list of available options in a range of integers.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None

Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**validation**(*x: Any*) → bool

Enure the entered text can be normalized.

**value**() → str

Return the ipywidget's value.

**class** euporie.core.comm.ipywidgets.**IpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *Comm*

A Comm object which represents ipython widgets.

**abstract create_view**(*parent: OutputParent*) → *CommView*

Abstract method for creating a view of the ipywidget.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**class** euporie.core.comm.ipywidgets.**LabelModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

A label ipwidget.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the Label widget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**LayoutIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

Base class for layout widgets with children.

**box_style**() → str

Convert the ipywidget box_style to a prompt_toolkit style string.

**buffers: Sequence[bytes]**

**abstract create_view**(*parent: OutputParent*) → *CommView*

Abstract method for creating a view of the ipywidget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**render_children**(*models: list[str]*, *parent: OutputParent*) → list[AnyContainer]

Create views for the child Comms in the layout.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**NumberTextBoxIpyWidgetComm**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *TextBoxIpyWidgetComm*

Base class for text-box ipywidgets with numerical values.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*
Create a new view of the numerical text-box ipywidget.

**data: dict[str, Any]**

**decr**(*button: Button*) → None
Decrement the widget's value by one step.

**default_rows = 1**

**incr**(*button: Button*) → None
Increment the widget's value by one step.

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*
Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → Any | None
Enure the selected value is permitted and the correct type.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None
Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer: Buffer*) → None
Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**validation**(*x: Any*) → bool
Enure the entered text can be normalized.

**value**() → str

Return the ipywidget's value.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

class euporie.core.comm.ipywidgets.**OutputModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

An Output ipywidget.

**add_output**(*json: dict[str, Any]*) → None

Add a new output to this widget.

**buffers: Sequence[bytes]**

**clear_output**(*wait: bool* = *False*) → None

Remove all outputs from this widget.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of this output ipywidget.

**data: dict[str, Any]**

**model_name = 'OutputModel'**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Modify the callbacks of a given message to add outputs to this ipywidget.

**set_state**(*key: str*, *value: JSONType*) → None

Send a comm_msg to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

class euporie.core.comm.ipywidgets.**ProgressIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

The base class for progress bar ipywidgets.

**bar_style**() → str

Convert the ipywidget bar_style to a prompt_toolkit style string.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the progress bar.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**RadioButtonsModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *SelectableIpyWidgetComm*

An ipywidget allowing an item to be selected using radio buttons.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the radio-buttons widget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_index**(*container:* SelectableWidget) → None

Send a `comm_message` updating the selected index when it changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**RangeSliderIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *SliderIpyWidgetComm*

Base class for range slider ipywidgets.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the slider ipywidget.

**data: dict[str, Any]**

**property indices: list[int]**

Return the first and last selected indices.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → Any

Convert the internal widget's value to one compatible with the ipywidget.

**abstract property options: list**

Abstract method to return a list of available options.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**set_value**(*slider:* Slider, *values: Any*) → None

Any float value is permitted - we might need to add an option.

**target_name = 'jupyter.widget'**

**update_value**(*slider:* SelectableWidget) → None

Send a `comm_message` updating the values when they change.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**SelectModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *SelectableIpyWidgetComm*

An ipywidget allowing a value to be selected from a list of options.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the select widget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_index**(*container:* SelectableWidget) → None

Send a `comm_message` updating the selected index when it changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**SelectMultipleModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

An ipywidget allowing one or more value to be selected from a list of options.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the multiple select widget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_index**(*container:* SelectableWidget) → None

Send a `comm_message` updating the selected index when it changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**SelectableIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

Base class for selectable ipywidgets.

**buffers: Sequence[bytes]**

**abstract create_view**(*parent: OutputParent*) → *CommView*

Abstract method for creating a view of the ipywidget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None
Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_index**(*container:* SelectableWidget) → None
Send a `comm_message` updating the selected index when it changes.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**SelectionRangeSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *SliderIpyWidgetComm*

A slider ipywidget where one or more of a list of options can be selected.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*
Create a new view of the slider ipywidget.

**data: dict[str, Any]**

**property indices: list[int]**
Return a list of the selected indices.

**new_view**(*parent: OutputParent*) → *CommView*
Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → Any
Convert the internal widget's value to one compatible with the ipywidget.

**property options: list[str]**
Return a list of the available options.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None
Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**set_value**(*slider:* Slider, *indices: tuple[int, int]*) → None
Set the selected indices on the selection slider when the value changes.

**target_name = 'jupyter.widget'**

**update_value**(*slider:* SelectableWidget) → None
Send a `comm_message` updating the selected indices when they change.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**SelectionSliderModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *SliderIpyWidgetComm*

A slider ipywidget where one of a list of options can be selected.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the slider ipywidget.

**data: dict[str, Any]**

**property indices: list[int]**

Return the selected index as a list.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → Any

Convert the internal widget's value to one compatible with the ipywidget.

**property options: list[str]**

Return a list of the available options.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**set_value**(*slider:* Slider, *index: int*) → None

Set the selected index on the selection slider when the value changes.

**target_name = 'jupyter.widget'**

**update_value**(*slider:* SelectableWidget) → None

Send a `comm_message` updating the selected index when it changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**SliderIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

Base class for slider ipywidgets.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the slider ipywidget.

**data: dict[str, Any]**

**property indices: list[Any]**

Return the selected index as a list.

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → Any

Convert the internal widget's value to one compatible with the ipywidget.

**abstract property options: list**

Abstract method to return a list of available options.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**set_value**(*slider:* Slider, *value: Any*) → None

Set the selected index when the ipywidget's selected value changes.

**target_name = 'jupyter.widget'**

**update_value**(*container:* SelectableWidget) → None

Send a `comm_message` updating the value when it changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**TabModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *LayoutIpyWidgetComm*

A tabbed layout ipywidget.

**box_style**() → str

Convert the ipywidget box_style to a prompt_toolkit style string.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the tabbed ipywidget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**render_children**(*models: list[str]*, *parent: OutputParent*) → list[AnyContainer]

Create views for the child Comms in the layout.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_index**(*container:* StackedSplit) → None
Send a `comm_message` updating the selected index when it changes.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**TextBoxIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

A mixin for ipywidgets which use text-box entry.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*
Create a new view of the text-box ipywidget.

**data: dict[str, Any]**

**default_rows = 1**

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*
Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → Any | None
Enure the selected value is permitted and the correct type.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None
Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None
Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**validation**(*x: Any*) → bool
Enure the entered text can be normalized.

**value**() → str
Return the ipywidget's value.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**TextModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *TextBoxIpyWidgetComm*

A text input widget.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the text-box ipywidget.

**data: dict[str, Any]**

**default_rows = 1**

**multiline = False**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → Any | None

Enure the selected value is permitted and the correct type.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None

Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**validation**(*x: Any*) → bool

Enure the entered text can be normalized.

**value**() → str

Return the ipywidget's value.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**TextareaModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *TextBoxIpyWidgetComm*

A text input widget.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the text-box ipywidget.

**data: dict[str, Any]**

**default_rows = 3**

**multiline = True**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → Any | None

Enure the selected value is permitted and the correct type.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_value**(*buffer:* Buffer) → None

Set the selected index when the ipywidget's entered value changes.

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**validation**(*x: Any*) → bool

Enure the entered text can be normalized.

**value**() → str

Return the ipywidget's value.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**ToggleButtonModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *ToggleableIpyWidgetComm*

A toggleable button ipywidget.

**buffers: Sequence[bytes]**

**button_style**() → str

Convert the ipywidget button_style to a prompt_toolkit style string.

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the toggle button ipywidget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → float | None

Cat the container's selected value to a bool if possible.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**text**() → str
Generate the button text, optionally including an icon if specified.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**value_changed**(*button:* ToggleableWidget) → None
Send a `comm_message` updating the value when it changes.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**ToggleButtonsModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

An ipywidget where a single value can be selected using togglable buttons.

**buffers: Sequence[bytes]**

**button_style**() → str
Convert the ipywidget button_style to a prompt_toolkit style string.

**create_view**(*parent: OutputParent*) → *CommView*
Create a new view of the toggle button widget.

**data: dict[str, Any]**

**get_label**(*index: int*) → AnyFormattedText
Return the label for each toggle button (optionally including the icon).

**new_view**(*parent: OutputParent*) → *CommView*
Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None
Handle incoming Comm update messages, updating the state and views.

**set_options**(*buttons:* ToggleButtons) → None
Set the list of selectable options in the toggle buttons.

**set_state**(*key: str*, *value: JSONType*) → None
Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_index**(*container:* SelectableWidget) → None
Set the selected index be sending a comm update message.

**update_views**(*changes: dict*) → None
Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**ToggleableIpyWidgetComm**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

Base class for toggleable ipywidgets.

**buffers: Sequence[bytes]**

**abstract create_view**(*parent: OutputParent*) → *CommView*

Abstract method for creating a view of the ipywidget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**normalize**(*x: Any*) → float | None

Cat the container's selected value to a bool if possible.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**value_changed**(*button: ToggleableWidget*) → None

Send a `comm_message` updating the value when it changes.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**UnimplementedModel**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *IpyWidgetComm*

An ipywidget used to represent unimplemented widgets.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**VBoxModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *BoxModel*

A vertical layout ipywidget.

**Split**

alias of *HSplit*

**box_style**() → str

Convert the ipywidget box_style to a prompt_toolkit style string.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the layout ipywidget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new *CommView* for this Comm.

**padding = 0**

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**render_children**(*models: list[str]*, *parent: OutputParent*) → list[AnyContainer]

Create views for the child Comms in the layout.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

**class** euporie.core.comm.ipywidgets.**ValidModel**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Bases: *ToggleableIpyWidgetComm*

A validity indicator ipywidget.

**buffers: Sequence[bytes]**

**create_view**(*parent: OutputParent*) → *CommView*

Create a new view of the validity ipywidget.

**data: dict[str, Any]**

**new_view**(*parent: OutputParent*) → *CommView*

Create and register a new `CommView` for this Comm.

**normalize**(*x: Any*) → float | None

Cat the container's selected value to a bool if possible.

**process_data**(*data: dict*, *buffers: Sequence[bytes]*) → None

Handle incoming Comm update messages, updating the state and views.

**set_state**(*key: str*, *value: JSONType*) → None

Send a `comm_msg` to the kernel with local state changes.

**target_name = 'jupyter.widget'**

**update_views**(*changes: dict*) → None

Update all the active views of this Comm.

**value_changed**(*button:* ToggleableWidget) → None

Send a `comm_message` updating the value when it changes.

**views: *WeakKeyDictionary*[*CommView*, OutputParent]**

euporie.core.comm.ipywidgets.**open_comm_ipywidgets**(*comm_container:* KernelTab, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*) → *IpyWidgetComm*

Create a new Comm for an `ipywidgets` widget.

The relevant widget model is selected based on the model name given in the `comm_open` message data.

**Parameters**

- **comm_container** – The notebook this Comm belongs to
- **comm_id** – The ID of the Comm
- **data** – The data field from the `comm_open` message
- **buffers** – The buffers field from the `comm_open` message

**Returns**

The initialized widget Comm object.

## euporie.core.comm.registry

Contain a registry of Comm target classes.

### Functions

| | |
|---|---|
| *open_comm*(comm_container, content, buffers) | Create a new object respsenting a Comm. |
| *open_comm_ipywidgets*(comm_container, ...) | Create a new Comm for an `ipywidgets` widget. |

### euporie.core.comm.registry.open_comm

euporie.core.comm.registry.**open_comm**(*comm_container: KernelTab*, *content: dict[str, Any]*, *buffers: Sequence[bytes]*) → *Comm*

Create a new object respsenting a Comm.

The class used to represent the Comm is determined by the "target_class" given in the `comm_open` message.

**Parameters**

- **comm_container** – The notebook this comm belongs to
- **content** – The content of the `comm_open` message
- **buffers** – A list of binary data buffers sent with the `comm_open` message

**Returns**

A class representing the comm

### euporie.core.comm.registry.open_comm_ipywidgets

euporie.core.comm.registry.**open_comm_ipywidgets**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*) → *IpyWidgetComm*

Create a new Comm for an `ipywidgets` widget.

The relevant widget model is selected based on the model name given in the `comm_open` message data.

**Parameters**

- **comm_container** – The notebook this Comm belongs to
- **comm_id** – The ID of the Comm
- **data** – The data field from the `comm_open` message
- **buffers** – The buffers field from the `comm_open` message

**Returns**

The initialized widget Comm object.

## Classes

| | |
|---|---|
| *UnimplementedComm*(comm_container, comm_id, ...) | Represent a Comm object which is not implemented in euporie.core. |

### euporie.core.comm.registry.UnimplementedComm

**class** euporie.core.comm.registry.**UnimplementedComm**(*comm_container: KernelTab*, *comm_id: str*, *data: dict*, *buffers: Sequence[bytes]*)

Represent a Comm object which is not implemented in euporie.core.

euporie.core.comm.registry.**open_comm**(*comm_container:* KernelTab, *content: dict[str, Any]*, *buffers: Sequence[bytes]*) → *Comm*

Create a new object respsenting a Comm.

The class used to represent the Comm is determined by the "target_class" given in the `comm_open` message.

**Parameters**

- **comm_container** – The notebook this comm belongs to
- **content** – The content of the `comm_open` message
- **buffers** – A list of binary data buffers sent with the `comm_open` message

**Returns**

A class representing the comm

### euporie.core.commands

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

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *cast*(typ, val) | Cast a value to a type. |
| *get_app*() | Get the current active (running) Application. |
| *get_cmd*(name) | Get a command from the centralized command system by name. |
| *isawaitable*(object) | Return true if object can be passed to an `await` expression. |
| *iscoroutinefunction*(obj) | Return true if the object is a coroutine function. |
| *parse_keys*(keys) | Pare a list of keys. |
| *signature*(obj, *[, follow_wrapped, globals, ...]) | Get a signature object for the passed callable. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |

### euporie.core.commands.add_cmd

euporie.core.commands.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.commands.cast

euporie.core.commands.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.commands.get_app

euporie.core.commands.**get_app**() → *Application*[Any]

Get the current active (running) Application. An *Application* is active during the Application.run_async() call.

We assume that there can only be one *Application* active at the same time. There is only one terminal window, with only one stdin and stdout. This makes the code significantly easier than passing around the *Application* everywhere.

If no *Application* is running, then return by default a DummyApplication. For practical reasons, we prefer to not raise an exception. This way, we don't have to check all over the place whether an actual *Application* was returned.

(For applications like pymux where we can have more than one *Application*, we'll use a work-around to handle that.)

### euporie.core.commands.get_cmd

euporie.core.commands.**get_cmd**(*name: str*) → *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.core.commands.isawaitable

euporie.core.commands.**isawaitable**(*object*)

Return true if object can be passed to an await expression.

### euporie.core.commands.iscoroutinefunction

euporie.core.commands.**iscoroutinefunction**(*obj*)

Return true if the object is a coroutine function.

Coroutine functions are normally defined with "async def" syntax, but may be marked via markcoroutinefunction.

### euporie.core.commands.parse_keys

euporie.core.commands.**parse_keys**(*keys: AnyKeys*) → list[tuple[str | *Keys*, ...]]

Pare a list of keys.

### euporie.core.commands.signature

euporie.core.commands.**signature**(*obj*, ***, *follow_wrapped=True*, *globals=None*, *locals=None*, *eval_str=False*)

Get a signature object for the passed callable.

### euporie.core.commands.to_filter

euporie.core.commands.**to_filter**(*bool_or_filter: Union[*Filter, *bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### Classes

| | |
|---|---|
| *Binding*(keys, ...], handler, filter, eager, ...) | Key binding: (key sequence + handler + filter). |
| *Command*(handler, *, filter, hidden, name, ...) | Wrap a function so it can be used as a key-binding or a menu item. |
| *KeyPressEvent*(key_processor_ref, arg, ...) | Key press event, delivered to key bindings. |

### euporie.core.commands.Binding

**class** euporie.core.commands.**Binding**(*keys: tuple[Keys | str, ...], handler: KeyHandlerCallable, filter: FilterOrBool = True, eager: FilterOrBool = False, is_global: FilterOrBool = False, save_before: Callable[[KeyPressEvent], bool] = <function Binding.<lambda>>, record_in_macro: FilterOrBool = True*)

Key binding: (key sequence + handler + filter). (Immutable binding class.)

**Parameters**

**record_in_macro** – When True, don't record this key binding when a macro is recorded.

### euporie.core.commands.Command

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

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

### euporie.core.commands.KeyPressEvent

**class** euporie.core.commands.**KeyPressEvent**(*key_processor_ref: ReferenceType[KeyProcessor]*, *arg: str | None*, *key_sequence: list[prompt_toolkit.key_binding.key_processor.KeyPress]*, *previous_key_sequence: list[prompt_toolkit.key_binding.key_processor.KeyPress]*, *is_repeat: bool*)

Key press event, delivered to key bindings.

**Parameters**

- **key_processor_ref** – Weak reference to the *KeyProcessor*.
- **arg** – Repetition argument.
- **key_sequence** – List of *KeyPress* instances.
- **previouskey_sequence** – Previous list of *KeyPress* instances.
- **is_repeat** – True when the previous event was delivered to the same handler.

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

Bases: object

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

**bind**(*key_bindings: KeyBindingsBase*, *keys: AnyKeys*) → 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_handler: KeyHandlerCallable**

Return a key handler for the command.

**key_str**() → str

Return a string representing the first registered key-binding.

**property menu: *MenuItem***

Return a menu item for the command.

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

Return a menu handler for the command.

**run**() → None

Run the command's handler.

euporie.core.commands.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

euporie.core.commands.**get_cmd**(*name: str*) → *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.core.completion

Contain the main class for a notebook file.

### Classes

| | |
|---|---|
| *CompleteEvent*([text_inserted, ...]) | Event that called the completer. |
| *Completer*() | Base class for completer implementations. |
| *Completion*(text[, start_position, display, ...]) | **param text** The new string that will be inserted into the document. |
| *DeduplicateCompleter*(completer) | Asynchronous wrapper around a completer that removes duplicates. |
| *KernelCompleter*(kernel) | A prompt_toolkit completer which provides completions from a Jupyter kernel. |
| *LspCompleter*(lsp, path) | A completer for documents using an LSP. |

### euporie.core.completion.CompleteEvent

**class** euporie.core.completion.**CompleteEvent**(*text_inserted: bool = False*, *completion_requested: bool = False*)

Event that called the completer.

**Parameters**

- **text_inserted** – When True, it means that completions are requested because of a text insert. (*Buffer.complete_while_typing*.)
- **completion_requested** – When True, it means that the user explicitly pressed the *Tab* key in order to view the completions.

These two flags can be used for instance to implement a completer that shows some completions when `Tab` has been pressed, but not automatically when the user presses a space. (Because of *complete_while_typing*.)

### euporie.core.completion.Completer

**class** euporie.core.completion.**Completer**

Base class for completer implementations.

### euporie.core.completion.Completion

**class** euporie.core.completion.**Completion**(*text: str*, *start_position: int = 0*, *display: AnyFormattedText | None = None*, *display_meta: AnyFormattedText | None = None*, *style: str = ''*, *selected_style: str = ''*)

**Parameters**

- **text** – The new string that will be inserted into the document.
- **start_position** – Position relative to the cursor_position where the new text will start. The text will be inserted between the start_position and the original cursor position.
- **display** – (optional string or formatted text) If the completion has to be displayed differently in the completion menu.
- **display_meta** – (Optional string or formatted text) Meta information about the completion, e.g. the path or source where it's coming from. This can also be a callable that returns a string.
- **style** – Style string.
- **selected_style** – Style string, used for a selected completion. This can override the *style* parameter.

### euporie.core.completion.DeduplicateCompleter

**class** euporie.core.completion.**DeduplicateCompleter**(*completer:* Completer)

Asynchronous wrapper around a completer that removes duplicates.

Only the first unique completions are kept. Completions are considered to be a duplicate if they result in the same document text when they would be applied.

### euporie.core.completion.KernelCompleter

**class** euporie.core.completion.**KernelCompleter**(*kernel:* Kernel)

A prompt_toolkit completer which provides completions from a Jupyter kernel.

### euporie.core.completion.LspCompleter

**class** euporie.core.completion.**LspCompleter**(*lsp:* LspClient, *path:* Path)

A completer for documents using an LSP.

**class** euporie.core.completion.**DeduplicateCompleter**(*completer:* Completer)

Bases: *Completer*

Asynchronous wrapper around a completer that removes duplicates.

Only the first unique completions are kept. Completions are considered to be a duplicate if they result in the same document text when they would be applied.

**get_completions**(*document:* Document, *complete_event:* CompleteEvent) → Iterable[*Completion*]

Do nothing as completions are retrieved asynchronously.

**async get_completions_async**(*document:* Document, *complete_event:* CompleteEvent) → AsyncGenerator[*Completion*, None]

Get completions from wrapped completer.

**class** euporie.core.completion.**KernelCompleter**(*kernel:* Kernel)

Bases: *Completer*

A prompt_toolkit completer which provides completions from a Jupyter kernel.

**get_completions**(*document:* Document, *complete_event:* CompleteEvent) → Iterable[*Completion*]

Do nothing as completions are retrieved asynchronously.

**async get_completions_async**(*document:* Document, *complete_event:* CompleteEvent) → AsyncGenerator[*Completion*, None]

Retrieve completions from a `Kernel`.

**class** euporie.core.completion.**LspCompleter**(*lsp:* LspClient, *path:* Path)

Bases: *Completer*

A completer for documents using an LSP.

**get_completions**(*document:* Document, *complete_event:* CompleteEvent) → Iterable[*Completion*]

Do nothing as completions are retrieved asynchronously.

**async get_completions_async**(*document:* Document, *complete_event:* CompleteEvent) → AsyncGenerator[*Completion*, None]

Retrieve completions from an LSP server.

### euporie.core.config

Define a configuration class for euporie.core.

### Functions

| | |
|---|---|
| add_cmd(**kwargs) | Add a command to the centralized command system. |
| add_setting(name, default, help_, description) | Register a new config item. |
| cast(typ, val) | Cast a value to a type. |
| get_cmd(name) | Get a command from the centralized command system by name. |
| literal_eval(node_or_string) | Evaluate an expression node or a string containing only a Python expression. |
| to_filter(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |
| user_config_dir([appname, appauthor, ...]) | **param appname** See *appname <platformdirs.api.PlatformDirsABC.appname>*. |

#### euporie.core.config.add_cmd

euporie.core.config.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

#### euporie.core.config.add_setting

euporie.core.config.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

#### euporie.core.config.cast

euporie.core.config.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.config.get_cmd

euporie.core.config.**get_cmd**(*name: str*) → *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.core.config.literal_eval

euporie.core.config.**literal_eval**(*node_or_string*)

Evaluate an expression node or a string containing only a Python expression. The string or node provided may only consist of the following Python literal structures: strings, bytes, numbers, tuples, lists, dicts, sets, booleans, and None.

Caution: A complex expression can overflow the C stack and cause a crash.

### euporie.core.config.to_filter

euporie.core.config.**to_filter**(*bool_or_filter: Union[*Filter*, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### euporie.core.config.user_config_dir

euporie.core.config.**user_config_dir**(*appname: str | None = None*, *appauthor: str | None | Literal[False] = None*, *version: str | None = None*, *roaming: bool = False*, *ensure_exists: bool = False*) → str

**Parameters**

- **appname** – See *appname <platformdirs.api.PlatformDirsABC.appname>*.
- **appauthor** – See *appauthor <platformdirs.api.PlatformDirsABC.appauthor>*.
- **version** – See *version <platformdirs.api.PlatformDirsABC.version>*.
- **roaming** – See *roaming <platformdirs.api.PlatformDirsABC.roaming>*.
- **ensure_exists** – See *ensure_exists <platformdirs.api.PlatformDirsABC.ensure_exists>*.

**Returns**
config directory tied to the user

### Classes

| | |
|---|---|
| *ArgumentParser*([prog, usage, description, ...]) | An argument parser which lexes and formats the help message before printing it. |
| *BooleanOptionalAction*(option_strings, *args, ...) | Action for boolean flags. |
| *ChainMap*(*maps) | A ChainMap groups multiple dicts (or other mappings) together to create a single, updateable view. |
| *Condition*(func) | Turn any callable into a Filter. |
| *Config*() | A configuration store. |
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *JSONEncoderPlus*(*[, skipkeys, ensure_ascii, ...]) | JSON encode class which encodes paths as strings. |
| *Path*(*args, **kwargs) | PurePath subclass that can make system calls. |
| *Protocol*() | Base class for protocol classes. |
| *Setting*(name, default, help_, description[, ...]) | A single configuration item. |
| *TextIO*() | Typed version of the return of open() in text mode. |
| *UPath*(*args[, protocol]) | |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.config.ArgumentParser

**class** euporie.core.config.**ArgumentParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*, *exit_on_error=True*)

An argument parser which lexes and formats the help message before printing it.

### euporie.core.config.BooleanOptionalAction

**class** euporie.core.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.core.config.ChainMap

**class** euporie.core.config.**ChainMap**(**maps*)

A ChainMap groups multiple dicts (or other mappings) together to create a single, updateable view.

The underlying mappings are stored in a list. That list is public and can be accessed or updated using the *maps* attribute. There is no other state.

Lookups search the underlying mappings successively until a key is found. In contrast, writes, updates, and deletions only operate on the first mapping.

### euporie.core.config.Condition

**class** euporie.core.config.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.config.Config

**class** euporie.core.config.**Config**

A configuration store.

### euporie.core.config.Event

**class** euporie.core.config.**Event**(*sender: _Sender*, *handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.config.JSONEncoderPlus

**class** euporie.core.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.

### euporie.core.config.Path

**class** euporie.core.config.**Path**(**args*, ***kwargs*)

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

### euporie.core.config.Protocol

**class** euporie.core.config.**Protocol**

Base class for protocol classes.

Protocol classes are defined as:

```
class Proto(Protocol):
    def meth(self) -> int:
        ...
```

Such classes are primarily used with static type checkers that recognize structural subtyping (static duck-typing).

For example:

```
class C:
    def meth(self) -> int:
        return 0

def func(x: Proto) -> int:
    return x.meth()

func(C())  # Passes static type check
```

See PEP 544 for details. Protocol classes decorated with @typing.runtime_checkable act as simple-minded runtime protocols that check only the presence of given attributes, ignoring their type signatures. Protocol classes can be generic, they are defined as:

```
class GenProto[T](Protocol):
    def meth(self) -> T:
        ...
```

### euporie.core.config.Setting

**class** euporie.core.config.**Setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *title: str | None = None*, *choices: list[Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*)

A single configuration item.

### euporie.core.config.TextIO

**class** euporie.core.config.**TextIO**

Typed version of the return of open() in text mode.

### euporie.core.config.UPath

**class** euporie.core.config.**UPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

### euporie.core.config.partial

**class** euporie.core.config.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.config.**ArgumentParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*, *exit_on_error=True*)

Bases: ArgumentParser

An argument parser which lexes and formats the help message before printing it.

**add_argument**(*dest*, ..., *name=value*, ...)

**add_argument**(*option_string*, *option_string*, ..., *name=value*, ...) → None

**add_argument_group**(**args*, ***kwargs*)

**add_mutually_exclusive_group**(***kwargs*)

**add_subparsers**(***kwargs*)

**convert_arg_line_to_args**(*arg_line*)

**error**(*message: string*)

Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

**exit**(*status=0*, *message=None*)

**format_help**()

**format_usage**()

**get_default**(*dest*)

**parse_args**(*args=None*, *namespace=None*)

**parse_intermixed_args**(*args=None*, *namespace=None*)

**parse_known_args**(*args=None*, *namespace=None*)

**parse_known_intermixed_args**(*args=None*, *namespace=None*)

**print_help**(*file=None*)

**print_usage**(*file=None*)

**register**(*registry_name*, *value*, *object*)

**set_defaults**(***kwargs*)

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

Bases: Action

Action for boolean flags.

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

**format_usage**() → str

Format the action string.

**Returns**
The formatted string.

**class** euporie.core.config.**Config**

Bases: object

A configuration store.

**conf_file_name = 'config.json'**

**filter**(*name: str*) → *Filter*

Return a Filter for a configuration item.

**get**(*name: str*, *default: Any = None*) → Any

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

**Parameters**

- **name** – The name of the attribute to access.
- **default** – The value to return if the name is not found

**Returns**
The configuration variable value.

**get_item**(*name: str*) → Any

Access a configuration item.

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

**Returns**
The configuration item.

**load**(*cls: type[ConfigurableApp]*) → None

Load the command line, environment, and user configuration.

**load_args**() → dict[str, Any]

Attempt to load configuration settings from commandline flags.

**load_config_file**() → dict[str, Any]

Attempt to load JSON configuration file.

**load_env**(*app_name: str = ''*) → dict[str, Any]

Attempt to load configuration settings from environment variables.

**load_parser**() → ArgumentParser

Construct an *ArgumentParser*.

**load_user**(*app_name: str = ''*) → dict[str, Any]

Attempt to load JSON configuration file.

**property schema: dict[str, Any]**

Return a JSON schema for the config.

```
settings: ClassVar[dict[str, Setting]] = {'accent_color': <Setting
accent_color='ansiblue'>, 'always_show_tab_bar': <Setting
always_show_tab_bar=False>, 'app': <Setting app='notebook'>, 'auth':
<Setting auth=True>, 'autocomplete': <Setting autocomplete=False>,
'autoformat': <Setting autoformat=False>, 'autoinspect': <Setting
autoinspect=False>, 'autosuggest': <Setting autosuggest=True>,
'background_character': <Setting background_character='·'>,
'background_pattern': <Setting background_pattern=2>, 'cell_start':
<Setting cell_start=None>, 'cell_stop': <Setting cell_stop=None>,
'client_keys': <Setting client_keys=['~/.ssh/authorized_keys']>,
'clipboard': <Setting clipboard='external'>, 'color_depth': <Setting
color_depth=None>, 'color_scheme': <Setting color_scheme='default'>,
'connection_file': <Setting connection_file=None>, 'cursor_blink':
<Setting cursor_blink=False>, 'custom_background_color': <Setting
custom_background_color='#073642'>, 'custom_foreground_color': <Setting
custom_foreground_color='#839496'>, 'edit_mode': <Setting
edit_mode='micro'>, 'enable_language_servers': <Setting
enable_language_servers=False>, 'expand': <Setting expand=False>,
'external_editor': <Setting external_editor=None>, 'files': <Setting
files=[]>, 'force_graphics': <Setting force_graphics=False>, 'formatters':
<Setting formatters=[]>, 'graphics': <Setting graphics=None>, 'host':
<Setting host=''>, 'host_keys': <Setting
host_keys=['/etc/ssh/ssh_host_ecdsa_key']>, 'kernel_name': <Setting
kernel_name='python3'>, 'key_bindings': <Setting key_bindings={}>,
'language_servers': <Setting language_servers={}>, 'line_numbers':
<Setting line_numbers=True>, 'log_config': <Setting log_config=None>,
'log_file': <Setting log_file=''>, 'log_level': <Setting
log_level='warning'>, 'max_notebook_width': <Setting
max_notebook_width=120>, 'max_stored_outputs': <Setting
max_stored_outputs=100>, 'mouse_support': <Setting mouse_support=None>,
'multiplexer_passthrough': <Setting multiplexer_passthrough=False>,
'output_file': <Setting output_file='-'>, 'page': <Setting page=False>,
'port': <Setting port=8022>, 'record_cell_timing': <Setting
record_cell_timing=False>, 'run': <Setting run=False>,
'run_after_external_edit': <Setting run_after_external_edit=False>,
'save': <Setting save=False>, 'save_widget_state': <Setting
save_widget_state=True>, 'set_cursor_shape': <Setting
set_cursor_shape=True>, 'show_cell_borders': <Setting
show_cell_borders=False>, 'show_file_icons': <Setting
show_file_icons=False>, 'show_filenames': <Setting show_filenames=False>,
'show_scroll_bar': <Setting show_scroll_bar=True>, 'show_shadows':
<Setting show_shadows=True>, 'show_side_bar': <Setting
show_side_bar=False>, 'show_status_bar': <Setting show_status_bar=True>,
'show_top_bar': <Setting show_top_bar=True>, 'syntax_theme': <Setting
syntax_theme='euporie'>, 'tab_mode': <Setting tab_mode='stack'>,
'tab_size': <Setting tab_size=4>, 'terminal_polling_interval': <Setting
terminal_polling_interval=0.0>, 'version': <Setting version=False>,
'wrap_cell_outputs': <Setting wrap_cell_outputs=False>}
```

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

Bases: JSONEncoder

JSON encode class which encodes paths as strings.

**default**(*o: Any*) → 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 = ': '**

**class** euporie.core.config.**Setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *title: str | None = None*, *choices: list[Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*)

Bases: object

A single configuration item.

**property menu: *MenuItem***

Return a menu item for the setting.

**property parser_args: tuple[list[str], dict[str, Any]]**

Return arguments for construction of an argparse.ArgumentParser.

**register_commands**() → None

Register commands to set this setting.

**property schema: dict[str, Any]**

Return a json schema property for the config item.

**toggle**() → None

Toggle the setting's value.

**property value: Any**

Return the current value.

euporie.core.config.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.convert

Sub-module concerned with the conversion of data formats.

### Modules

| | |
|---|---|
| *euporie.core.convert.datum* | Convert display data between formats. |
| *euporie.core.convert.formats* | Contain various data format conversion functions. |
| *euporie.core.convert.mime* | Contain main format conversion function. |
| *euporie.core.convert.registry* | Contain main format conversion function. |
| *euporie.core.convert.utils* | Utility functions for format converters. |

### euporie.core.convert.datum

Convert display data between formats.

### Functions

| | |
|---|---|
| *get_app*() | Get the current active (running) Application. |
| *get_loop*() | Create or return the conversion IO loop. |
| *to_plain_text*(value) | Turn any kind of formatted text back into plain text. |

### euporie.core.convert.datum.get_app

euporie.core.convert.datum.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.convert.datum.get_loop

euporie.core.convert.datum.**get_loop**() → AbstractEventLoop

Create or return the conversion IO loop.

The loop will be running on a separate thread.

### euporie.core.convert.datum.to_plain_text

euporie.core.convert.datum.**to_plain_text**(*value: AnyFormattedText*) → str

Turn any kind of formatted text back into plain text.

### Classes

| | |
|---|---|
| *Datum*(data, *args, **kwargs) | Class for storing and converting display data. |
| *Generic*() | Abstract base class for generic types. |
| *PilImage* | alias of `Image` |
| *ReferenceType* | |
| *Size*(rows, columns) | |
| *TypeVar* | Type variable. |
| *WeakValueDictionary*([other]) | Mapping class that references values weakly. |
| *WindowAlign*(value[, names, module, ...]) | Alignment of the Window content. |
| *finalize*(obj, func, /, *args, **kwargs) | Class for finalization of weakrefable objects |
| *ref* | alias of *ReferenceType* |

### euporie.core.convert.datum.Datum

**class** euporie.core.convert.datum.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Class for storing and converting display data.

### euporie.core.convert.datum.Generic

**class** euporie.core.convert.datum.**Generic**

Abstract base class for generic types.

On Python 3.12 and newer, generic classes implicitly inherit from Generic when they declare a parameter list after the class's name:

```
class Mapping[KT, VT]:
    def __getitem__(self, key: KT) -> VT:
        ...
    # Etc.
```

On older versions of Python, however, generic classes have to explicitly inherit from Generic.

After a class has been declared to be generic, it can then be used as follows:

```
def lookup_name[KT, VT](mapping: Mapping[KT, VT], key: KT, default: VT) -> VT:
    try:
        return mapping[key]
    except KeyError:
        return default
```

### euporie.core.convert.datum.PilImage

euporie.core.convert.datum.**PilImage**

alias of Image

### euporie.core.convert.datum.ReferenceType

**class** euporie.core.convert.datum.**ReferenceType**

### euporie.core.convert.datum.Size

**class** euporie.core.convert.datum.**Size**(*rows*, *columns*)

### euporie.core.convert.datum.TypeVar

**class** euporie.core.convert.datum.**TypeVar**

Type variable.

The preferred way to construct a type variable is via the dedicated syntax for generic functions, classes, and type aliases:

```
class Sequence[T]:  # T is a TypeVar
    ...
```

This syntax can also be used to create bound and constrained type variables:

```
# S is a TypeVar bound to str
class StrSequence[S: str]:
    ...

# A is a TypeVar constrained to str or bytes
class StrOrBytesSequence[A: (str, bytes)]:
    ...
```

However, if desired, reusable type variables can also be constructed manually, like so:

```
T = TypeVar('T')  # Can be anything
S = TypeVar('S', bound=str)  # Can be any subtype of str
A = TypeVar('A', str, bytes)  # Must be exactly str or bytes
```

Type variables exist primarily for the benefit of static type checkers. They serve as the parameters for generic types as well as for generic function and type alias definitions.

The variance of type variables is inferred by type checkers when they are created through the type parameter syntax and when `infer_variance=True` is passed. Manually created type variables may be explicitly marked covariant or contravariant by passing `covariant=True` or `contravariant=True`. By default, manually created type variables are invariant. See PEP 484 and PEP 695 for more details.

### euporie.core.convert.datum.WeakValueDictionary

**class** euporie.core.convert.datum.**WeakValueDictionary**(*other=()*, */*, ***kw*)

Mapping class that references values weakly.

Entries in the dictionary will be discarded when no strong reference to the value exists anymore

### euporie.core.convert.datum.WindowAlign

**class** euporie.core.convert.datum.**WindowAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of the Window content.

Note that this is different from *HorizontalAlign* and *VerticalAlign*, which are used for the alignment of the child containers in respectively *VSplit* and *HSplit*.

### euporie.core.convert.datum.finalize

**class** euporie.core.convert.datum.**finalize**(*obj*, *func*, */*, **args*, ***kwargs*)

Class for finalization of weakrefable objects

finalize(obj, func, *args, **kwargs) returns a callable finalizer object which will be called when obj is garbage collected. The first time the finalizer is called it evaluates func(*arg, **kwargs) and returns the result. After this the finalizer is dead, and calling it just returns None.

When the program exits any remaining finalizers for which the atexit attribute is true will be run in reverse order of creation. By default atexit is true.

### euporie.core.convert.datum.ref

euporie.core.convert.datum.**ref**

alias of *ReferenceType*

**class** euporie.core.convert.datum.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Bases: *Generic*[T]

Class for storing and converting display data.

**add_size**(*size: tuple[int, int]* | prompt_toolkit.data_structures.Size) → str

Store a size for a :py:class`Datum`.

**cell_size**() → Any

Get cell width and aspect synchronously.

**async cell_size_async**() → tuple[int, float]

Get the cell width and aspect ratio of the displayable data.

**Returns**

**A tuple of the data's width in terminal columns and its aspect ratio, when** converted to a image.

**convert**(*to: str*, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → Any

Convert between formats.

**async convert_async**(*to: str*, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*, *bbox:* DiInt | *None = None*) → Any

Perform conversion asynchronously, caching the result.

**static get_hash**(*data: Any*) → str

Calculate a hash of data.

**classmethod get_size**(*key: str*) → tuple[*euporie.core.convert.datum.Datum*, *prompt_toolkit.data_structures.Size*] | None

Retrieve a `Datum` and it's size by its key.

**property hash: str**

Return a hash of the *Datum*'s data.

**pixel_size**() → Any

Get data dimensions synchronously.

**async pixel_size_async**() → tuple[int | None, int | None]

Get the dimensions of displayable data in pixels.

Foreground and background color are set at this point if they are available, as data conversion outputs are cached and re-used.

**Returns**

**A tuple of the data's width in terminal columns and its aspect ratio, when** converted to a image.

**property root: *Datum***

Retrieve the source datum of any conversion outputs.

**to_bytes**() → bytes

Cast the data to bytes.

euporie.core.convert.datum.**get_loop**() → AbstractEventLoop

Create or return the conversion IO loop.

The loop will be running on a separate thread.

### euporie.core.convert.formats

Contain various data format conversion functions.

They are grouped into sub-modules based on output format.

### Modules

| | |
|---|---|
| *euporie.core.convert.formats.ansi* | Contain functions which convert data to formatted ansi text. |
| *euporie.core.convert.formats.base64* | Contain functions which convert data to base64 format. |
| *euporie.core.convert.formats.common* | Contain functions which can be used to convert data to multiple formats. |
| *euporie.core.convert.formats.ft* | Contain functions which convert data to formatted text. |
| *euporie.core.convert.formats.html* | Contain functions which convert data to html format. |
| *euporie.core.convert.formats.jpeg* | Contain functions which convert data to jpeg format. |
| *euporie.core.convert.formats.markdown* | Contain functions which convert data to markdown format. |
| *euporie.core.convert.formats.pdf* | Contain function which convert data to pdf format. |
| *euporie.core.convert.formats.pil* | Contain functions which convert data to PIL format. |
| *euporie.core.convert.formats.png* | Contain functions which convert data to png format. |
| *euporie.core.convert.formats.rich* | Contain function which convert data to rich format. |
| *euporie.core.convert.formats.sixel* | Contain function which convert data to sixel format. |
| *euporie.core.convert.formats.svg* | Contain function which convert data to SVG format. |

### euporie.core.convert.formats.ansi

Contain functions which convert data to formatted ansi text.

### Functions

| | |
|---|---|
| call_subproc(data, cmd[, use_tempfile, suffix]) | Call the command as a subprocess and return it's output as bytes. |
| ceil(x, /) | Return the ceiling of x as an Integral. |
| chafa_convert_cmd(output_format, datum[, ...]) | Convert image data to ANSI text using **chafa**. |
| chafa_convert_py(output_format, datum[, ...]) | Convert image data to ANSI text using ::*chafa.py*. |
| command_exists(*cmds) | Verify a list of external commands exist on the system. |
| get_app() | Get the current active (running) Application. |
| have_modules(*modules) | Verify a list of python modules are importable. |
| html_to_ansi_elinks(datum[, cols, rows, fg, ...]) | Convert HTML text to formatted ANSI using **elinks**. |
| html_to_ansi_links(datum[, cols, rows, fg, ...]) | Convert HTML text to formatted ANSI using **links**. |
| html_to_ansi_lynx(datum[, cols, rows, fg, ...]) | Convert HTML text to formatted ANSI using **lynx**. |
| html_to_ansi_py_htmlparser(datum[, cols, ...]) | Convert HTML tables to ANSI text using `HTMLParser`. |
| html_to_ansi_w3m(datum[, cols, rows, fg, ...]) | Convert HTML text to formatted ANSI using **w3m**. |
| image_to_ansi_catimg(datum[, cols, rows, ...]) | Convert image data to ANSI text using **catimg**. |
| image_to_ansi_icat(datum[, cols, rows, fg, ...]) | Convert image data to ANSI text using **icat**. |
| image_to_ansi_jp2a(datum[, cols, rows, fg, ...]) | Convert image data to ANSI text using **jp2a**. |
| image_to_ansi_timg(datum[, cols, rows, fg, ...]) | Convert image data to ANSI text using **timg**. |
| image_to_ansi_tiv(datum[, cols, rows, fg, ...]) | Convert image data to ANSI text using **tiv**. |
| image_to_ansi_viu(datum[, cols, rows, fg, ...]) | Convert image data to ANSI text using **viu**. |
| latex_to_ansi_py_flatlatex(datum[, cols, ...]) | Convert LaTeX to ANSI using `flatlatex`. |
| latex_to_ansi_py_pylatexenc(datum[, cols, ...]) | Convert LaTeX to ANSI using `pylatexenc`. |
| latex_to_ansi_py_sympy(datum[, cols, rows, ...]) | Convert LaTeX to ANSI using `sympy`. |
| latex_to_ansi_utftex(datum[, cols, rows, ...]) | Render LaTeX maths as unicode. |
| pil_to_ansi_py_img2unicode(datum[, cols, ...]) | Convert a PIL image to ANSI text using `img2unicode`. |
| pil_to_ansi_py_timg(datum[, cols, rows, fg, ...]) | Convert a PIL image to ANSI text using `timg`. |
| png_to_ansi_img2txt(datum[, cols, rows, fg, ...]) | Convert PNG data to ANSI text using **img2txt**. |
| png_to_ansi_py_placeholder(datum[, cols, ...]) | Draw placeholder ANSI text. |
| register(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |
| rich_to_ansi_py(datum[, cols, rows, fg, bg, ...]) | Convert rich objects to formatted ANSI text. |
| set_background(image[, bg]) | Remove the alpha channel from an image and set the background colour. |

### euporie.core.convert.formats.ansi.call_subproc

**async** euporie.core.convert.formats.ansi.**call_subproc**(*data: str | bytes*, *cmd: list[Any]*, *use_tempfile: bool = False*, *suffix: str = ''*) → 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
- **suffix** – Suffix for the temporary file name

**Returns**
The data printed to standard out by the subprocess.

### euporie.core.convert.formats.ansi.ceil

euporie.core.convert.formats.ansi.**ceil**(*x*, */*)
Return the ceiling of x as an Integral.

This is the smallest integer >= x.

### euporie.core.convert.formats.ansi.chafa_convert_cmd

**async** euporie.core.convert.formats.ansi.**chafa_convert_cmd**(*output_format: str*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to ANSI text using **chafa**.

### euporie.core.convert.formats.ansi.chafa_convert_py

**async** euporie.core.convert.formats.ansi.**chafa_convert_py**(*output_format: Literal['symbols', 'sixels', 'kitty', 'iterm2']*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to ANSI text using ::*chafa.py*.

### euporie.core.convert.formats.ansi.command_exists

euporie.core.convert.formats.ansi.**command_exists**(**cmds: str*) → *Filter*
Verify a list of external commands exist on the system.

### euporie.core.convert.formats.ansi.get_app

euporie.core.convert.formats.ansi.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.convert.formats.ansi.have_modules

euporie.core.convert.formats.ansi.**have_modules**(**modules: str*) → *Filter*

Verify a list of python modules are importable.

### euporie.core.convert.formats.ansi.html_to_ansi_elinks

**async** euporie.core.convert.formats.ansi.**html_to_ansi_elinks**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str

Convert HTML text to formatted ANSI using **elinks**.

### euporie.core.convert.formats.ansi.html_to_ansi_links

**async** euporie.core.convert.formats.ansi.**html_to_ansi_links**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str

Convert HTML text to formatted ANSI using **links**.

### euporie.core.convert.formats.ansi.html_to_ansi_lynx

**async** euporie.core.convert.formats.ansi.**html_to_ansi_lynx**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str

Convert HTML text to formatted ANSI using **lynx**.

### euporie.core.convert.formats.ansi.html_to_ansi_py_htmlparser

**async** euporie.core.convert.formats.ansi.**html_to_ansi_py_htmlparser**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert HTML tables to ANSI text using `HTMLParser`.

### euporie.core.convert.formats.ansi.html_to_ansi_w3m

**async** euporie.core.convert.formats.ansi.**html_to_ansi_w3m**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert HTML text to formatted ANSI using **`w3m`**.

### euporie.core.convert.formats.ansi.image_to_ansi_catimg

**async** euporie.core.convert.formats.ansi.**image_to_ansi_catimg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **`catimg`**.

### euporie.core.convert.formats.ansi.image_to_ansi_icat

**async** euporie.core.convert.formats.ansi.**image_to_ansi_icat**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **`icat`**.

### euporie.core.convert.formats.ansi.image_to_ansi_jp2a

**async** euporie.core.convert.formats.ansi.**image_to_ansi_jp2a**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **jp2a**.

### euporie.core.convert.formats.ansi.image_to_ansi_timg

**async** euporie.core.convert.formats.ansi.**image_to_ansi_timg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **timg**.

### euporie.core.convert.formats.ansi.image_to_ansi_tiv

**async** euporie.core.convert.formats.ansi.**image_to_ansi_tiv**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **tiv**.

### euporie.core.convert.formats.ansi.image_to_ansi_viu

**async** euporie.core.convert.formats.ansi.**image_to_ansi_viu**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **viu**.

### euporie.core.convert.formats.ansi.latex_to_ansi_py_flatlatex

**async** euporie.core.convert.formats.ansi.**latex_to_ansi_py_flatlatex**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert LaTeX to ANSI using `flatlatex`.

### euporie.core.convert.formats.ansi.latex_to_ansi_py_pylatexenc

**async** euporie.core.convert.formats.ansi.**latex_to_ansi_py_pylatexenc**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert LaTeX to ANSI using `pylatexenc`.

### euporie.core.convert.formats.ansi.latex_to_ansi_py_sympy

**async** euporie.core.convert.formats.ansi.**latex_to_ansi_py_sympy**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert LaTeX to ANSI using `sympy`.

### euporie.core.convert.formats.ansi.latex_to_ansi_utftex

**async** euporie.core.convert.formats.ansi.**latex_to_ansi_utftex**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Render LaTeX maths as unicode.

### euporie.core.convert.formats.ansi.pil_to_ansi_py_img2unicode

**async** euporie.core.convert.formats.ansi.**pil_to_ansi_py_img2unicode**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert a PIL image to ANSI text using `img2unicode`.

### euporie.core.convert.formats.ansi.pil_to_ansi_py_timg

**async** euporie.core.convert.formats.ansi.**pil_to_ansi_py_timg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert a PIL image to ANSI text using `timg`.

### euporie.core.convert.formats.ansi.png_to_ansi_img2txt

**async** euporie.core.convert.formats.ansi.**png_to_ansi_img2txt**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert PNG data to ANSI text using **`img2txt`**.

### euporie.core.convert.formats.ansi.png_to_ansi_py_placeholder

**async** euporie.core.convert.formats.ansi.**png_to_ansi_py_placeholder**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Draw placeholder ANSI text.

### euporie.core.convert.formats.ansi.register

euporie.core.convert.formats.ansi.**register**(*from_:* *Iterable[str]* | *str*, *to:* *str*, *filter_:* *FilterOrBool* = *True*, *weight:* *int* = *1*) → Callable

Add a converter to the centralized format conversion system.

### euporie.core.convert.formats.ansi.rich_to_ansi_py

**async** euporie.core.convert.formats.ansi.**rich_to_ansi_py**(*datum:* Datum, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → str

Convert rich objects to formatted ANSI text.

### euporie.core.convert.formats.ansi.set_background

euporie.core.convert.formats.ansi.**set_background**(*image: PilImage*, *bg: str | None = None*) → PilImage

Remove the alpha channel from an image and set the background colour.

### Classes

| | |
|---|---|
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.convert.formats.ansi.partial

**class** euporie.core.convert.formats.ansi.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**async** euporie.core.convert.formats.ansi.**html_to_ansi_elinks**(*datum:* Datum, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → str

Convert HTML text to formatted ANSI using **elinks**.

**async** euporie.core.convert.formats.ansi.**html_to_ansi_links**(*datum:* Datum, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → str

Convert HTML text to formatted ANSI using **links**.

**async** euporie.core.convert.formats.ansi.**html_to_ansi_lynx**(*datum:* Datum, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → str

Convert HTML text to formatted ANSI using **lynx**.

**async** euporie.core.convert.formats.ansi.**html_to_ansi_py_htmlparser**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert HTML tables to ANSI text using `HTMLParser`.

**async** euporie.core.convert.formats.ansi.**html_to_ansi_w3m**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert HTML text to formatted ANSI using **w3m**.

**async** euporie.core.convert.formats.ansi.**image_to_ansi_catimg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **catimg**.

**async** euporie.core.convert.formats.ansi.**image_to_ansi_icat**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **icat**.

**async** euporie.core.convert.formats.ansi.**image_to_ansi_jp2a**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **jp2a**.

**async** euporie.core.convert.formats.ansi.**image_to_ansi_timg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **timg**.

**async** euporie.core.convert.formats.ansi.**image_to_ansi_tiv**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **tiv**.

**async** euporie.core.convert.formats.ansi.**image_to_ansi_viu**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert image data to ANSI text using **viu**.

**async** euporie.core.convert.formats.ansi.**latex_to_ansi_py_flatlatex**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert LaTeX to ANSI using `flatlatex`.

**async** euporie.core.convert.formats.ansi.**latex_to_ansi_py_pylatexenc**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert LaTeX to ANSI using `pylatexenc`.

**async** euporie.core.convert.formats.ansi.**latex_to_ansi_py_sympy**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert LaTeX to ANSI using `sympy`.

**async** euporie.core.convert.formats.ansi.**latex_to_ansi_utftex**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Render LaTeX maths as unicode.

**async** euporie.core.convert.formats.ansi.**pil_to_ansi_py_img2unicode**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert a PIL image to ANSI text using `img2unicode`.

**async** euporie.core.convert.formats.ansi.**pil_to_ansi_py_timg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert a PIL image to ANSI text using `timg`.

**async** euporie.core.convert.formats.ansi.**png_to_ansi_img2txt**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert PNG data to ANSI text using **`img2txt`**.

**async** euporie.core.convert.formats.ansi.**png_to_ansi_py_placeholder**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Draw placeholder ANSI text.

**async** euporie.core.convert.formats.ansi.**rich_to_ansi_py**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert rich objects to formatted ANSI text.

### euporie.core.convert.formats.base64

Contain functions which convert data to base64 format.

### Functions

| | |
|---|---|
| *bytes_to_base64_py*(datum[, cols, rows, fg, ...]) | Convert bytes to base64 encoded data. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |

### euporie.core.convert.formats.base64.bytes_to_base64_py

**async** euporie.core.convert.formats.base64.**bytes_to_base64_py**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str

Convert bytes to base64 encoded data.

### euporie.core.convert.formats.base64.register

euporie.core.convert.formats.base64.**register**(*from_: Iterable[str]* | *str*, *to: str*, *filter_: FilterOrBool* = *True*, *weight: int* = *1*) → Callable

Add a converter to the centralized format conversion system.

**async** euporie.core.convert.formats.base64.**bytes_to_base64_py**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str

Convert bytes to base64 encoded data.

### euporie.core.convert.formats.common

Contain functions which can be used to convert data to multiple formats.

#### Functions

| | |
|---|---|
| *base64_to_bytes_py*(datum[, cols, rows, fg, ...]) | Convert base64 encoded data to bytes. |
| *call_subproc*(data, cmd[, use_tempfile, suffix]) | Call the command as a subprocess and return it's output as bytes. |
| *chafa_convert_cmd*(output_format, datum[, ...]) | Convert image data to ANSI text using **chafa**. |
| *chafa_convert_py*(output_format, datum[, ...]) | Convert image data to ANSI text using ::*chafa.py*. |
| *get_app*() | Get the current active (running) Application. |
| *imagemagick_convert*(output_format, datum[, ...]) | Convert image data to PNG bytes using `imagemagick`. |

### euporie.core.convert.formats.common.base64_to_bytes_py

**async** euporie.core.convert.formats.common.**base64_to_bytes_py**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → bytes

Convert base64 encoded data to bytes.

### euporie.core.convert.formats.common.call_subproc

**async** euporie.core.convert.formats.common.**call_subproc**(*data: str* | *bytes*, *cmd: list[Any]*, *use_tempfile: bool* = *False*, *suffix: str* = '') → 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
- **suffix** – Suffix for the temporary file name

**Returns**

The data printed to standard out by the subprocess.

### euporie.core.convert.formats.common.chafa_convert_cmd

**async** euporie.core.convert.formats.common.**chafa_convert_cmd**(*output_format: str*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to ANSI text using **chafa**.

### euporie.core.convert.formats.common.chafa_convert_py

**async** euporie.core.convert.formats.common.**chafa_convert_py**(*output_format: Literal['symbols', 'sixels', 'kitty', 'iterm2']*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to ANSI text using ::*chafa.py*.

### euporie.core.convert.formats.common.get_app

euporie.core.convert.formats.common.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.convert.formats.common.imagemagick_convert

**async** euporie.core.convert.formats.common.**imagemagick_convert**(*output_format: str*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to PNG bytes using `imagemagick`.

**async** euporie.core.convert.formats.common.**base64_to_bytes_py**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → bytes

Convert base64 encoded data to bytes.

**async** euporie.core.convert.formats.common.**chafa_convert_cmd**(*output_format: str*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to ANSI text using **`chafa`**.

**async** euporie.core.convert.formats.common.**chafa_convert_py**(*output_format: Literal['symbols', 'sixels', 'kitty', 'iterm2']*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to ANSI text using ::*chafa.py*.

**async** euporie.core.convert.formats.common.**imagemagick_convert**(*output_format: str*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to PNG bytes using `imagemagick`.

### euporie.core.convert.formats.ft

Contain functions which convert data to formatted text.

### Functions

| | |
|---|---|
| *ansi_to_ft*(datum[, cols, rows, fg, bg, ...]) | Convert ANSI text to formatted text, lexing & formatting automatically. |
| *detect_lexer*([text, path, language]) | Detect the pygments lexer for a file. |
| *html_to_ft*(datum[, cols, rows, fg, bg, extend]) | Convert HTML to formatted text. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |
| *strip_one_trailing_newline*(ft) | Remove up to one trailing new-line character from formatted text. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |

### euporie.core.convert.formats.ft.ansi_to_ft

**async** euporie.core.convert.formats.ft.**ansi_to_ft**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*, *lex:* *bool* = *False*) → StyleAndTextTuples

Convert ANSI text to formatted text, lexing & formatting automatically.

### euporie.core.convert.formats.ft.detect_lexer

euporie.core.convert.formats.ft.**detect_lexer**(*text:* *str* = '', *path:* Path | *None* = *None*, *language:* *str* = '') → PygmentsLexerCls | None

Detect the pygments lexer for a file.

### euporie.core.convert.formats.ft.html_to_ft

**async** euporie.core.convert.formats.ft.**html_to_ft**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → StyleAndTextTuples

Convert HTML to formatted text.

### euporie.core.convert.formats.ft.register

euporie.core.convert.formats.ft.**register**(*from_: Iterable[str] | str*, *to: str*, *filter_: FilterOrBool = True*, *weight: int = 1*) → Callable

Add a converter to the centralized format conversion system.

### euporie.core.convert.formats.ft.strip_one_trailing_newline

euporie.core.convert.formats.ft.**strip_one_trailing_newline**(*ft: StyleAndTextTuples*) → StyleAndTextTuples

Remove up to one trailing new-line character from formatted text.

### euporie.core.convert.formats.ft.to_formatted_text

euporie.core.convert.formats.ft.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

## Classes

| | |
|---|---|
| *ANSI*(value[, tab_size]) | Convert ANSI text into formatted text, preserving all control sequences. |
| *SimpleCache*([maxsize]) | Very simple cache that discards the oldest item when the cache size is exceeded. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.convert.formats.ft.ANSI

**class** euporie.core.convert.formats.ft.**ANSI**(*value: str*, *tab_size: int = 8*)

Convert ANSI text into formatted text, preserving all control sequences.

### euporie.core.convert.formats.ft.SimpleCache

**class** euporie.core.convert.formats.ft.**SimpleCache**(*maxsize: int = 8*)

Very simple cache that discards the oldest item when the cache size is exceeded.

**Parameters**

**maxsize** – Maximum size of the cache. (Don't make it too big.)

### euporie.core.convert.formats.ft.partial

**class** euporie.core.convert.formats.ft.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**async** euporie.core.convert.formats.ft.**ansi_to_ft**(*datum: Datum*, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*, *lex: bool = False*) → StyleAndTextTuples

Convert ANSI text to formatted text, lexing & formatting automatically.

**async** euporie.core.convert.formats.ft.**html_to_ft**(*datum: Datum*, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → StyleAndTextTuples

Convert HTML to formatted text.

### euporie.core.convert.formats.html

Contain functions which convert data to html format.

### Functions

| | |
|---|---|
| *amsmath_plugin*(md, *[, renderer]) | Parses TeX math equations, without any surrounding delimiters, only for top-level amsmath environments: |
| *detect_lexer*([text, path, language]) | Detect the pygments lexer for a file. |
| *dollarmath_plugin*(md, *[, allow_labels, ...]) | Plugin for parsing dollar enclosed math, e.g. |
| *get_app*() | Get the current active (running) Application. |
| *highlight*(code, lexer, formatter[, outfile]) | This is the most high-level highlighting function. |
| *markdown_to_html_markdown_it*(datum[, cols, ...]) | Convert markdown to HTML using `markdownit_py`. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |
| *texmath_plugin*(md[, delimiters, macros]) | Plugin ported from markdown-it-texmath. |

### euporie.core.convert.formats.html.amsmath_plugin

euporie.core.convert.formats.html.**amsmath_plugin**(*md:* MarkdownIt, *, *renderer:* *Optional[Callable[[str], str]]* = *None*)

Parses TeX math equations, without any surrounding delimiters, only for top-level amsmath environments:

```
\begin{gather*}
a_1=b_1+c_1\\
a_2=b_2+c_2-d_2+e_2
\end{gather*}
```

**Parameters**

**renderer** – Function to render content, by default escapes HTML

### euporie.core.convert.formats.html.detect_lexer

euporie.core.convert.formats.html.**detect_lexer**(*text:* *str* = '', *path:* Path | *None* = *None*, *language:* *str* = '') → PygmentsLexerCls | None

Detect the pygments lexer for a file.

### euporie.core.convert.formats.html.dollarmath_plugin

euporie.core.convert.formats.html.**dollarmath_plugin**(*md:* MarkdownIt, *, *allow_labels:* *bool* = *True*, *allow_space:* *bool* = *True*, *allow_digits:* *bool* = *True*, *double_inline:* *bool* = *False*, *label_normalizer:* *Optional[Callable[[str], str]]* = *None*, *renderer:* *Optional[Callable[[str, Dict[str, Any]], str]]* = *None*, *label_renderer:* *Optional[Callable[[str], str]]* = *None*) → None

Plugin for parsing dollar enclosed math, e.g. inline: `$a=1$`, block: `$$b=2$$`

This is an improved version of `texmath`; it is more performant, and handles `\` escaping properly and allows for more configuration.

**Parameters**

- **allow_labels** – Capture math blocks with label suffix, e.g. `$$a=1$$ (eq1)`
- **allow_space** – Parse inline math when there is space after/before the opening/closing `$`, e.g. `$ a $`
- **allow_digits** – Parse inline math when there is a digit before/after the opening/closing `$`, e.g. `1$` or `$2`. This is useful when also using currency.
- **double_inline** – Search for double-dollar math within inline contexts
- **label_normalizer** – Function to normalize the label, by default replaces whitespace with `-`
- **renderer** – Function to render content: *(str, {"display_mode": bool})* -> *str*, by default escapes HTML

- **label_renderer** – Function to render labels, by default creates anchor

### euporie.core.convert.formats.html.get_app

euporie.core.convert.formats.html.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.convert.formats.html.highlight

euporie.core.convert.formats.html.**highlight**(*code*, *lexer*, *formatter*, *outfile=None*)

This is the most high-level highlighting function. It combines *lex* and *format* in one function.

### euporie.core.convert.formats.html.markdown_to_html_markdown_it

**async** euporie.core.convert.formats.html.**markdown_to_html_markdown_it**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert markdown to HTML using `markdownit_py`.

### euporie.core.convert.formats.html.register

euporie.core.convert.formats.html.**register**(*from_:* *Iterable[str]* | *str*, *to:* *str*, *filter_:* *FilterOrBool* = *True*, *weight:* *int* = *1*) → Callable

Add a converter to the centralized format conversion system.

### euporie.core.convert.formats.html.texmath_plugin

euporie.core.convert.formats.html.**texmath_plugin**(*md:* MarkdownIt, *delimiters='dollars'*, *macros:* *Optional[dict]* = *None*)

Plugin ported from markdown-it-texmath.

It parses TeX math equations set inside opening and closing delimiters:

```
$\alpha = \frac{1}{2}$
```

**Parameters**

**delimiters** – one of: brackets, dollars, gitlab, julia, kramdown

### Classes

| | |
|---|---|
| *HtmlFormatter*(**options) | Format tokens as HTML 4  tags. |
| *MarkdownIt*(config, options_update, *, ...) | |
| *MarkdownParser*(config, options_update, *, ...) | Subclas the markdown parser to allow file: URIs. |

### euporie.core.convert.formats.html.HtmlFormatter

**class** euporie.core.convert.formats.html.**HtmlFormatter**(***options*)

Format tokens as HTML 4  tags. By default, the content is enclosed in a <pre> tag, itself wrapped in a  tag (but see the *nowrap* option). The 's CSS class can be set by the *cssclass* option.

If the *linenos* option is set to "table", the <pre> is additionally wrapped inside a <table> which has one row and two cells: one containing the line numbers and one containing the code. Example:

```
<div class="highlight" >
<table><tr>
  <td class="linenos" title="click to toggle"
    onclick="with (this.firstChild.style)
             { display = (display == '') ? 'none' : '' }">
    <pre>1
    2</pre>
  </td>
  <td class="code">
    <pre><span class="Ke">def </span><span class="NaFu">foo</span>(bar):
      <span class="Ke">pass</span>
    </pre>
  </td>
</tr></table></div>
```

(whitespace added to improve clarity).

A list of lines can be specified using the *hl_lines* option to make these lines highlighted (as of Pygments 0.11).

With the *full* option, a complete HTML 4 document is output, including the style definitions inside a <style> tag, or in a separate file if the *cssfile* option is given.

When *tagsfile* is set to the path of a ctags index file, it is used to generate hyperlinks from names to their definition. You must enable *lineanchors* and run ctags with the *-n* option for this to work. The *python-ctags* module from PyPI must be installed to use this feature; otherwise a *RuntimeError* will be raised.

The *get_style_defs(arg='')* method of a *HtmlFormatter* returns a string containing CSS rules for the CSS classes used by the formatter. The argument *arg* can be used to specify additional CSS selectors that are prepended to the classes. A call *fmter.get_style_defs('td .code')* would result in the following CSS classes:

```
td .code .kw { font-weight: bold; color: #00FF00 }
td .code .cm { color: #999999 }
...
```

If you have Pygments 0.6 or higher, you can also pass a list or tuple to the *get_style_defs()* method to request multiple prefixes for the tokens:

```
formatter.get_style_defs(['div.syntax pre', 'pre.syntax'])
```

The output would then look like this:

```
div.syntax pre .kw,
pre.syntax .kw { font-weight: bold; color: #00FF00 }
div.syntax pre .cm,
pre.syntax .cm { color: #999999 }
...
```

Additional options accepted:

***nowrap***
If set to `True`, don't add a `<pre>` and a `<div>` tag around the tokens. This disables most other options (default: `False`).

***full***
Tells the formatter to output a "full" document, i.e. a complete self-contained document (default: `False`).

***title***
If *full* is true, the title that should be used to caption the document (default: `''`).

***style***
The style to use, can be a string or a Style subclass (default: `'default'`). This option has no effect if the *cssfile* and *noclobber_cssfile* option are given and the file specified in *cssfile* exists.

***noclasses***
If set to true, token `<span>` tags (as well as line number elements) will not use CSS classes, but inline styles. This is not recommended for larger pieces of code since it increases output size by quite a bit (default: `False`).

***classprefix***
Since the token types use relatively short class names, they may clash with some of your own class names. In this case you can use the *classprefix* option to give a string to prepend to all Pygments-generated CSS class names for token types. Note that this option also affects the output of *get_style_defs()*.

***cssclass***
CSS class for the wrapping `<div>` tag (default: `'highlight'`). If you set this option, the default selector for *get_style_defs()* will be this class.

New in version 0.9: If you select the `'table'` line numbers, the wrapping table will have a CSS class of this string plus `'table'`, the default is accordingly `'highlighttable'`.

***cssstyles***
Inline CSS styles for the wrapping `<div>` tag (default: `''`).

***prestyles***
Inline CSS styles for the `<pre>` tag (default: `''`).

New in version 0.11.

***cssfile***
If the *full* option is true and this option is given, it must be the name of an external file. If the filename does not include an absolute path, the file's path will be assumed to be relative to the main output file's path, if the latter can be found. The stylesheet is then written to this file instead of the HTML file.

New in version 0.6.

***noclobber_cssfile***
If *cssfile* is given and the specified file exists, the css file will not be overwritten. This allows the use of the *full* option in combination with a user specified css file. Default is `False`.

New in version 1.1.

***linenos***
If set to `'table'`, output line numbers as a table with two cells, one containing the line numbers, the other

the whole code. This is copy-and-paste-friendly, but may cause alignment problems with some browsers or fonts. If set to `'inline'`, the line numbers will be integrated in the `<pre>` tag that contains the code (that setting is *new in Pygments 0.8*).

For compatibility with Pygments 0.7 and earlier, every true value except `'inline'` means the same as `'table'` (in particular, that means also `True`).

The default value is `False`, which means no line numbers at all.

**Note:** with the default ("table") line number mechanism, the line numbers and code can have different line heights in Internet Explorer unless you give the enclosing `<pre>` tags an explicit `line-height` CSS property (you get the default line spacing with `line-height: 125%`).

***hl_lines***
Specify a list of lines to be highlighted. The line numbers are always relative to the input (i.e. the first line is line 1) and are independent of *linenostart*.

New in version 0.11.

***linenostart***
The line number for the first line (default: `1`).

***linenostep***
If set to a number n > 1, only every nth line number is printed.

***linenospecial***
If set to a number n > 0, every nth line number is given the CSS class `"special"` (default: `0`).

***nobackground***
If set to `True`, the formatter won't output the background color for the wrapping element (this automatically defaults to `False` when there is no wrapping element [eg: no argument for the *get_syntax_defs* method given]) (default: `False`).

New in version 0.6.

***lineseparator***
This string is output between lines of code. It defaults to `"\n"`, which is enough to break a line inside `<pre>` tags, but you can e.g. set it to `"<br>"` to get HTML line breaks.

New in version 0.7.

***lineanchors***
If set to a nonempty string, e.g. `foo`, the formatter will wrap each output line in an anchor tag with an `id` (and *name*) of `foo-linenumber`. This allows easy linking to certain lines.

New in version 0.9.

***linespans***
If set to a nonempty string, e.g. `foo`, the formatter will wrap each output line in a span tag with an `id` of `foo-linenumber`. This allows easy access to lines via javascript.

New in version 1.6.

***anchorlinenos***
If set to *True*, will wrap line numbers in <a> tags. Used in combination with *linenos* and *lineanchors*.

***tagsfile***
If set to the path of a ctags file, wrap names in anchor tags that link to their definitions. *lineanchors* should be used, and the tags file should specify line numbers (see the *-n* option to ctags). The tags file is assumed to be encoded in UTF-8.

New in version 1.6.

***tagurlformat***
A string formatting pattern used to generate links to ctags definitions. Available variables are *%(path)s*, *%(fname)s* and *%(fext)s*. Defaults to an empty string, resulting in just *#prefix-number* links.

New in version 1.6.

***filename***
A string used to generate a filename when rendering `<pre>` blocks, for example if displaying source code. If *linenos* is set to `'table'` then the filename will be rendered in an initial row containing a single *<th>* which spans both columns.

New in version 2.1.

***wrapcode***
Wrap the code inside `<pre>` blocks using `<code>`, as recommended by the HTML5 specification.

New in version 2.4.

***debug_token_types***
Add `title` attributes to all token `<span>` tags that show the name of the token.

New in version 2.10.

**Subclassing the HTML formatter**

New in version 0.7.

The HTML formatter is now built in a way that allows easy subclassing, thus customizing the output HTML code. The *format()* method calls *self._format_lines()* which returns a generator that yields tuples of `(1, line)`, where the `1` indicates that the `line` is a line of the formatted source code.

If the *nowrap* option is set, the generator is the iterated over and the resulting HTML is output.

Otherwise, *format()* calls *self.wrap()*, which wraps the generator with other generators. These may add some HTML code to the one generated by *_format_lines()*, either by modifying the lines generated by the latter, then yielding them again with `(1, line)`, and/or by yielding other HTML code before or after the lines, with `(0, html)`. The distinction between source lines and other code makes it possible to wrap the generator multiple times.

The default *wrap()* implementation adds a `<div>` and a `<pre>` tag.

A custom *HtmlFormatter* subclass could look like this:

```
class CodeHtmlFormatter(HtmlFormatter):

    def wrap(self, source, *, include_div):
        return self._wrap_code(source)

    def _wrap_code(self, source):
        yield 0, '<code>'
        for i, t in source:
            if i == 1:
                # it's a line of formatted code
                t += '<br>'
            yield i, t
        yield 0, '</code>'
```

This results in wrapping the formatted lines with a `<code>` tag, where the source lines are broken using `<br>` tags.

After calling *wrap()*, the *format()* method also adds the “line numbers” and/or “full document” wrappers if the respective options are set. Then, all HTML yielded by the wrapped generator is output.

### euporie.core.convert.formats.html.MarkdownIt

**class** euporie.core.convert.formats.html.**MarkdownIt**(*config: str | collections.abc.Mapping = 'commonmark', options_update: collections.abc.Mapping | None = None, *, renderer_cls: ~collections.abc.Callable[[~markdown_it.main.MarkdownIt], ~markdown_it.renderer.RendererProtocol] = <class 'markdown_it.renderer.RendererHTML'>*)

### euporie.core.convert.formats.html.MarkdownParser

**class** euporie.core.convert.formats.html.**MarkdownParser**(*config: str | collections.abc.Mapping = 'commonmark', options_update: collections.abc.Mapping | None = None, *, renderer_cls: ~collections.abc.Callable[[~markdown_it.main.MarkdownIt], ~markdown_it.renderer.RendererProtocol] = <class 'markdown_it.renderer.RendererHTML'>*)

Subclas the markdown parser to allow `file:` URIs.

**class** euporie.core.convert.formats.html.**MarkdownParser**(*config: str | collections.abc.Mapping = 'commonmark', options_update: collections.abc.Mapping | None = None, *, renderer_cls: ~collections.abc.Callable[[~markdown_it.main.MarkdownIt], ~markdown_it.renderer.RendererProtocol] = <class 'markdown_it.renderer.RendererHTML'>*)

Bases: *MarkdownIt*

Subclas the markdown parser to allow `file:` URIs.

**add_render_rule**(*name: str*, *function: Callable*, *fmt: str = 'html'*) → None

Add a rule for rendering a particular Token type.

Only applied when `renderer.__output__ == fmt`

**configure**(*presets: str* | collections.abc.Mapping, *options_update:* collections.abc.Mapping | *None = None*) → *MarkdownIt*

Batch load of all options and component settings. This is an internal method, and you probably will not need it. But if you will - see available presets and data structure [here](https://github.com/markdown-it/markdown-it/tree/master/lib/presets)

We strongly recommend to use presets instead of direct config loads. That will give better compatibility with next versions.

**disable**(*names: str | collections.abc.Iterable[str]*, *ignoreInvalid: bool = False*) → *MarkdownIt*

The same as [[MarkdownIt.enable]], but turn specified rules off. (chainable)

**Parameters**

- **names** – rule name or list of rule names to disable.
- **ignoreInvalid** – set *true* to ignore errors when rule not found.

**enable**(*names: str | collections.abc.Iterable[str]*, *ignoreInvalid: bool = False*) → *MarkdownIt*

Enable list or rules. (chainable)

**Parameters**

- **names** – rule name or list of rule names to enable.
- **ignoreInvalid** – set *true* to ignore errors when rule not found.

It will automatically find appropriate components, containing rules with given names. If rule not found, and *ignoreInvalid* not set - throws exception.

Example:

```
md = MarkdownIt().enable(['sub', 'sup']).disable('smartquotes')
```

**get_active_rules**() → dict[str, list[str]]

Return the names of all active rules.

**get_all_rules**() → dict[str, list[str]]

Return the names of all active rules.

**normalizeLink**(*url: str*) → str

Normalize destination URLs in links

```
[label]:   destination   'title'
        ^^^^^^^^^^^
```

**normalizeLinkText**(*link: str*) → str

Normalize autolink content

```
<destination>
~~~~~~~~~~~~
```

**parse**(*src: str*, *env: collections.abc.MutableMapping | None = None*) → list[markdown_it.token.Token]

Parse the source string to a token stream

**Parameters**

- **src** – source string
- **env** – environment sandbox

Parse input string and return list of block tokens (special token type "inline" will contain list of inline tokens).

*env* is used to pass data between "distributed" rules and return additional metadata like reference info, needed for the renderer. It also can be used to inject data in specific cases. Usually, you will be ok to pass *{}*, and then pass updated object to renderer.

**parseInline**(*src: str*, *env: collections.abc.MutableMapping | None = None*) → list[markdown_it.token.Token]

The same as [[MarkdownIt.parse]] but skip all block rules.

**Parameters**

- **src** – source string
- **env** – environment sandbox

It returns the block tokens list with the single *inline* element, containing parsed inline tokens in *children* property. Also updates *env* object.

**render**(*src: str*, *env: collections.abc.MutableMapping | None = None*) → Any

Render markdown string into html. It does all magic for you :).

**Parameters**

- **src** – source string
- **env** – environment sandbox

**Returns**

The output of the loaded renderer

*env* can be used to inject additional metadata (*{}* by default). But you will not need it with high probability. See also comment in [[MarkdownIt.parse]].

**renderInline**(*src: str*, *env: collections.abc.MutableMapping | None = None*) → Any

Similar to [[MarkdownIt.render]] but for single paragraph content.

**Parameters**

- **src** – source string
- **env** – environment sandbox

Similar to [[MarkdownIt.render]] but for single paragraph content. Result will NOT be wrapped into <*p*> tags.

**reset_rules**() → Generator[None, None, None]

A context manager, that will reset the current enabled rules on exit.

**set**(*options: MutableMapping*) → None

Set parser options (in the same format as in constructor). Probably, you will never need it, but you can change options after constructor call.

__Note:__ To achieve the best possible performance, don't modify a *markdown-it* instance options on the fly. If you need multiple configurations it's best to create multiple instances and initialize each with separate config.

**use**(*plugin: Callable*, **params*, ***options*) → *MarkdownIt*

Load specified plugin with given params into current parser instance. (chainable)

It's just a sugar to call *plugin(md, params)* with curring.

Example:

```
def func(tokens, idx):
    tokens[idx].content = tokens[idx].content.replace('foo', 'bar')
md = MarkdownIt().use(plugin, 'foo_replace', 'text', func)
```

**validateLink**(*url: str*) → bool

Allow all link URIs.

**async** euporie.core.convert.formats.html.**markdown_to_html_markdown_it**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert markdown to HTML using `markdownit_py`.

### euporie.core.convert.formats.jpeg

Contain functions which convert data to jpeg format.

### Functions

| | |
|---|---|
| *base64_to_bytes_py*(datum[, cols, rows, fg, ...]) | Convert base64 encoded data to bytes. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |

### euporie.core.convert.formats.jpeg.base64_to_bytes_py

**async** euporie.core.convert.formats.jpeg.**base64_to_bytes_py**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → bytes

Convert base64 encoded data to bytes.

### euporie.core.convert.formats.jpeg.register

euporie.core.convert.formats.jpeg.**register**(*from_: Iterable[str]* | *str*, *to: str*, *filter_: FilterOrBool* = *True*, *weight: int* = *1*) → Callable

Add a converter to the centralized format conversion system.

### euporie.core.convert.formats.markdown

Contain functions which convert data to markdown format.

### Functions

| | |
|---|---|
| *have_modules*(*modules) | Verify a list of python modules are importable. |
| *html_to_markdown_py_html2text*(datum[, cols, ...]) | Convert HTML to markdown tables using `html2text`. |
| *html_to_markdown_py_mtable*(datum[, cols, ...]) | Convert HTML tables to markdown tables using `mtable`. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |

### euporie.core.convert.formats.markdown.have_modules

`euporie.core.convert.formats.markdown.`**`have_modules`**(**modules: str*) → *Filter*

Verify a list of python modules are importable.

### euporie.core.convert.formats.markdown.html_to_markdown_py_html2text

**async** `euporie.core.convert.formats.markdown.`**`html_to_markdown_py_html2text`**(*datum: Datum*, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → str

Convert HTML to markdown tables using `html2text`.

### euporie.core.convert.formats.markdown.html_to_markdown_py_mtable

**async** euporie.core.convert.formats.markdown.**html_to_markdown_py_mtable**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str

Convert HTML tables to markdown tables using `mtable`.

### euporie.core.convert.formats.markdown.register

euporie.core.convert.formats.markdown.**register**(*from_: Iterable[str]* | *str*, *to: str*, *filter_: FilterOrBool = True*, *weight: int = 1*) → Callable

Add a converter to the centralized format conversion system.

**async** euporie.core.convert.formats.markdown.**html_to_markdown_py_html2text**(*datum: Datum*, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → str

Convert HTML to markdown tables using `html2text`.

**async** euporie.core.convert.formats.markdown.**html_to_markdown_py_mtable**(*datum: Datum*, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → str

Convert HTML tables to markdown tables using `mtable`.

### euporie.core.convert.formats.pdf

Contain function which convert data to pdf format.

#### Functions

| | |
|---|---|
| *base64_to_bytes_py*(datum[, cols, rows, fg, ...]) | Convert base64 encoded data to bytes. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |

### euporie.core.convert.formats.pdf.base64_to_bytes_py

**async** euporie.core.convert.formats.pdf.**base64_to_bytes_py**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → bytes

Convert base64 encoded data to bytes.

### euporie.core.convert.formats.pdf.register

euporie.core.convert.formats.pdf.**register**(*from_:* *Iterable[str]* | *str*, *to:* *str*, *filter_:* *FilterOrBool* = *True*, *weight:* *int* = *1*) → Callable

Add a converter to the centralized format conversion system.

### euporie.core.convert.formats.pil

Contain functions which convert data to PIL format.

#### Functions

| | |
|---|---|
| *have_modules*(*modules) | Verify a list of python modules are importable. |
| *png_to_pil_py*(datum[, cols, rows, fg, bg, ...]) | Convert PNG to a pillow image using `PIL`. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |
| *set_background*(image[, bg]) | Remove the alpha channel from an image and set the background colour. |

### euporie.core.convert.formats.pil.have_modules

euporie.core.convert.formats.pil.**have_modules**(**modules: str*) → *Filter*

Verify a list of python modules are importable.

### euporie.core.convert.formats.pil.png_to_pil_py

**async** euporie.core.convert.formats.pil.**png_to_pil_py**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → PilImage

Convert PNG to a pillow image using `PIL`.

### euporie.core.convert.formats.pil.register

euporie.core.convert.formats.pil.**register**(*from_: Iterable[str]* | *str*, *to: str*, *filter_: FilterOrBool* = *True*, *weight: int* = *1*) → Callable

Add a converter to the centralized format conversion system.

### euporie.core.convert.formats.pil.set_background

euporie.core.convert.formats.pil.**set_background**(*image: PilImage*, *bg: str* | *None* = *None*) → PilImage

Remove the alpha channel from an image and set the background colour.

**async** euporie.core.convert.formats.pil.**png_to_pil_py**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → PilImage

Convert PNG to a pillow image using `PIL`.

euporie.core.convert.formats.pil.**set_background**(*image: PilImage*, *bg: str* | *None* = *None*) → PilImage

Remove the alpha channel from an image and set the background colour.

### euporie.core.convert.formats.png

Contain functions which convert data to png format.

### Functions

| | |
|---|---|
| *base64_to_bytes_py*(datum[, cols, rows, fg, ...]) | Convert base64 encoded data to bytes. |
| *command_exists*(*cmds) | Verify a list of external commands exist on the system. |
| *have_modules*(*modules) | Verify a list of python modules are importable. |
| *imagemagick_convert*(output_format, datum[, ...]) | Convert image data to PNG bytes using `imagemagick`. |
| *latex_to_png_dvipng*(datum[, cols, rows, fg, ...]) | Render LaTeX as a png image using **`dvipng`**. |
| *latex_to_png_py_mpl*(datum[, cols, rows, fg, ...]) | Render LaTeX as a png image using **:py:module:`matplotlib`**. |
| *pil_to_png_py_pil*(datum[, cols, rows, fg, ...]) | Convert a pillow image to sixels `teimpy`. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |
| *svg_to_png_py_cairosvg*(datum[, cols, rows, ...]) | Convert SVG to PNG using `cairosvg`. |

#### euporie.core.convert.formats.png.base64_to_bytes_py

**async** euporie.core.convert.formats.png.**base64_to_bytes_py**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → bytes

Convert base64 encoded data to bytes.

#### euporie.core.convert.formats.png.command_exists

euporie.core.convert.formats.png.**command_exists**(**cmds:* *str*) → *Filter*

Verify a list of external commands exist on the system.

#### euporie.core.convert.formats.png.have_modules

euporie.core.convert.formats.png.**have_modules**(**modules:* *str*) → *Filter*

Verify a list of python modules are importable.

#### euporie.core.convert.formats.png.imagemagick_convert

**async** euporie.core.convert.formats.png.**imagemagick_convert**(*output_format:* *str*, *datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str | bytes

Convert image data to PNG bytes using `imagemagick`.

### euporie.core.convert.formats.png.latex_to_png_dvipng

**async** euporie.core.convert.formats.png.**latex_to_png_dvipng**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*, *timeout:* *int* = *2*) → bytes | None

Render LaTeX as a png image using **dvipng**.

Borrowed from IPython.

### euporie.core.convert.formats.png.latex_to_png_py_mpl

**async** euporie.core.convert.formats.png.**latex_to_png_py_mpl**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → bytes

Render LaTeX as a png image using **:py:module:`matplotlib`**.

Borrowed from IPython.

### euporie.core.convert.formats.png.pil_to_png_py_pil

**async** euporie.core.convert.formats.png.**pil_to_png_py_pil**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → bytes

Convert a pillow image to sixels `teimpy`.

### euporie.core.convert.formats.png.register

euporie.core.convert.formats.png.**register**(*from_:* *Iterable[str]* | *str*, *to:* *str*, *filter_:* *FilterOrBool* = *True*, *weight:* *int* = *1*) → Callable

Add a converter to the centralized format conversion system.

### euporie.core.convert.formats.png.svg_to_png_py_cairosvg

**async** euporie.core.convert.formats.png.**svg_to_png_py_cairosvg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert SVG to PNG using `cairosvg`.

### Classes

| | |
|---|---|
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.convert.formats.png.partial

**class** euporie.core.convert.formats.png.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**async** euporie.core.convert.formats.png.**latex_to_png_dvipng**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*, *timeout:* *int* = *2*) → bytes | None

Render LaTeX as a png image using **dvipng**.

Borrowed from IPython.

**async** euporie.core.convert.formats.png.**latex_to_png_py_mpl**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → bytes

Render LaTeX as a png image using **:py:module:`matplotlib`**.

Borrowed from IPython.

**async** euporie.core.convert.formats.png.**pil_to_png_py_pil**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → bytes

Convert a pillow image to sixels `teimpy`.

**async** euporie.core.convert.formats.png.**svg_to_png_py_cairosvg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert SVG to PNG using `cairosvg`.

### euporie.core.convert.formats.rich

Contain function which convert data to rich format.

### Functions

| | |
|---|---|
| *have_modules*(*modules) | Verify a list of python modules are importable. |
| *markdown_to_rich_py*(datum[, cols, rows, fg, ...]) | Convert base64 encoded data to bytes. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |

### euporie.core.convert.formats.rich.have_modules

euporie.core.convert.formats.rich.**have_modules**(**modules: str*) → *Filter*

Verify a list of python modules are importable.

### euporie.core.convert.formats.rich.markdown_to_rich_py

**async** euporie.core.convert.formats.rich.**markdown_to_rich_py**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → Markdown

Convert base64 encoded data to bytes.

### euporie.core.convert.formats.rich.register

euporie.core.convert.formats.rich.**register**(*from_: Iterable[str]* | *str*, *to: str*, *filter_: FilterOrBool* = *True*, *weight: int* = *1*) → Callable

Add a converter to the centralized format conversion system.

**async** euporie.core.convert.formats.rich.**markdown_to_rich_py**(*datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → Markdown

Convert base64 encoded data to bytes.

### euporie.core.convert.formats.sixel

Contain function which convert data to sixel format.

### Functions

| | |
|---|---|
| *call_subproc*(data, cmd[, use_tempfile, suffix]) | Call the command as a subprocess and return it's output as bytes. |
| *chafa_convert_cmd*(output_format, datum[, ...]) | Convert image data to ANSI text using **chafa**. |
| *chafa_convert_py*(output_format, datum[, ...]) | Convert image data to ANSI text using ::*chafa.py*. |
| *command_exists*(*cmds) | Verify a list of external commands exist on the system. |
| *get_app*() | Get the current active (running) Application. |
| *have_modules*(*modules) | Verify a list of python modules are importable. |
| *imagemagick_convert*(output_format, datum[, ...]) | Convert image data to PNG bytes using `imagemagick`. |
| *pil_to_sixel_py_teimpy*(datum[, cols, rows, ...]) | Convert a pillow image to sixels `teimpy`. |
| *pil_to_sixel_py_timg*(datum[, cols, rows, ...]) | Convert a pillow image to sixels `timg`. |
| *png_to_sixel_img2sixel*(datum[, cols, rows, ...]) | Convert PNG data to sixels **img2sixel**. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |

### euporie.core.convert.formats.sixel.call_subproc

**async** euporie.core.convert.formats.sixel.**call_subproc**(*data: str | bytes*, *cmd: list[Any]*, *use_tempfile: bool = False*, *suffix: str = ''*) → 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
- **suffix** – Suffix for the temporary file name

**Returns**

The data printed to standard out by the subprocess.

### euporie.core.convert.formats.sixel.chafa_convert_cmd

**async** euporie.core.convert.formats.sixel.**chafa_convert_cmd**(*output_format: str*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to ANSI text using **chafa**.

### euporie.core.convert.formats.sixel.chafa_convert_py

**async** euporie.core.convert.formats.sixel.**chafa_convert_py**(*output_format: Literal['symbols', 'sixels', 'kitty', 'iterm2']*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to ANSI text using ::*chafa.py*.

### euporie.core.convert.formats.sixel.command_exists

euporie.core.convert.formats.sixel.**command_exists**(**cmds: str*) → *Filter*

Verify a list of external commands exist on the system.

### euporie.core.convert.formats.sixel.get_app

euporie.core.convert.formats.sixel.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.convert.formats.sixel.have_modules

euporie.core.convert.formats.sixel.**have_modules**(**modules: str*) → *Filter*

Verify a list of python modules are importable.

### euporie.core.convert.formats.sixel.imagemagick_convert

**async** euporie.core.convert.formats.sixel.**imagemagick_convert**(*output_format: str*, *datum:* Datum, *cols: int* | *None* = *None*, *rows: int* | *None* = *None*, *fg: str* | *None* = *None*, *bg: str* | *None* = *None*, *extend: bool* = *True*) → str | bytes

Convert image data to PNG bytes using `imagemagick`.

### euporie.core.convert.formats.sixel.pil_to_sixel_py_teimpy

**async** euporie.core.convert.formats.sixel.**pil_to_sixel_py_teimpy**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert a pillow image to sixels `teimpy`.

### euporie.core.convert.formats.sixel.pil_to_sixel_py_timg

**async** euporie.core.convert.formats.sixel.**pil_to_sixel_py_timg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert a pillow image to sixels `timg`.

### euporie.core.convert.formats.sixel.png_to_sixel_img2sixel

**async** euporie.core.convert.formats.sixel.**png_to_sixel_img2sixel**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert PNG data to sixels **`img2sixel`**.

### euporie.core.convert.formats.sixel.register

euporie.core.convert.formats.sixel.**register**(*from_:* *Iterable[str]* | *str*, *to:* *str*, *filter_:* *FilterOrBool* = *True*, *weight:* *int* = *1*) → Callable

Add a converter to the centralized format conversion system.

### Classes

| | |
|---|---|
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.convert.formats.sixel.partial

**class** euporie.core.convert.formats.sixel.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**async** euporie.core.convert.formats.sixel.**pil_to_sixel_py_teimpy**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert a pillow image to sixels `teimpy`.

**async** euporie.core.convert.formats.sixel.**pil_to_sixel_py_timg**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert a pillow image to sixels `timg`.

**async** euporie.core.convert.formats.sixel.**png_to_sixel_img2sixel**(*datum:* Datum, *cols:* *int* | *None* = *None*, *rows:* *int* | *None* = *None*, *fg:* *str* | *None* = *None*, *bg:* *str* | *None* = *None*, *extend:* *bool* = *True*) → str

Convert PNG data to sixels **`img2sixel`**.

### euporie.core.convert.formats.svg

Contain function which convert data to SVG format.

## Functions

| | |
|---|---|
| *have_modules*(*modules) | Verify a list of python modules are importable. |
| *latex_to_svg_py_ziamath*(datum[, cols, rows, ...]) | Convert LaTeX to SVG using `ziamath`. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |

### euporie.core.convert.formats.svg.have_modules

euporie.core.convert.formats.svg.**have_modules**(**modules: str*) → *Filter*

Verify a list of python modules are importable.

### euporie.core.convert.formats.svg.latex_to_svg_py_ziamath

**async** euporie.core.convert.formats.svg.**latex_to_svg_py_ziamath**(*datum:* Datum, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → str

Convert LaTeX to SVG using `ziamath`.

### euporie.core.convert.formats.svg.register

euporie.core.convert.formats.svg.**register**(*from_: Iterable[str] | str*, *to: str*, *filter_: FilterOrBool = True*, *weight: int = 1*) → Callable

Add a converter to the centralized format conversion system.

**async** euporie.core.convert.formats.svg.**latex_to_svg_py_ziamath**(*datum:* Datum, *cols: int | None = None*, *rows: int | None = None*, *fg: str | None = None*, *bg: str | None = None*, *extend: bool = True*) → str

Convert LaTeX to SVG using `ziamath`.

### euporie.core.convert.mime

Contain main format conversion function.

### Functions

| | |
|---|---|
| *get_format*(path[, default]) | Attempt to guess the format of a path. |
| *get_mime*(path) | Attempt to determine the mime-type of a path. |
| *lru_cache*([maxsize, typed]) | Least-recently-used cache decorator. |

### euporie.core.convert.mime.get_format

euporie.core.convert.mime.**get_format**(*path:* Path | *str*, *default: str* = '') → str

Attempt to guess the format of a path.

### euporie.core.convert.mime.get_mime

euporie.core.convert.mime.**get_mime**(*path:* Path | *str*) → str | None

Attempt to determine the mime-type of a path.

### euporie.core.convert.mime.lru_cache

euporie.core.convert.mime.**lru_cache**(*maxsize=128*, *typed=False*)

Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

### Classes

| | |
|---|---|
| *HTTPPath*(*args[, protocol]) | |
| *UPath*(*args[, protocol]) | |
| *UPathStatResult*(stat_result_seq[, info_dict]) | A stat_result compatible class wrapping fsspec info dicts. |

### euporie.core.convert.mime.HTTPPath

**class** euporie.core.convert.mime.**HTTPPath**(**args*, *protocol: str* | *None* = *None*, ***storage_options: Any*)

### euporie.core.convert.mime.UPath

**class** euporie.core.convert.mime.**UPath**(**args*, *protocol: str* | *None* = *None*, ***storage_options: Any*)

### euporie.core.convert.mime.UPathStatResult

**class** euporie.core.convert.mime.**UPathStatResult**(*stat_result_seq: Sequence[int]*, *info_dict: Optional[Mapping[str, Any]] = None*)

A stat_result compatible class wrapping fsspec info dicts.

**Note: It is unlikely that you will ever have to instantiate**
this class directly. If you want to convert and info dict, use: *UPathStatResult.from_info(info)*

**This object may be accessed either as a tuple of**
(mode, ino, dev, nlink, uid, gid, size, atime, mtime, ctime)

or via the attributes st_mode, st_ino, st_dev, st_nlink, st_uid, and so on.

There's an additional method *as_info()* for accessing the info dict. This is useful to access additional information provided by the file system implementation, that's not covered by the stat_result tuple.

euporie.core.convert.mime.**get_format**(*path:* Path | *str*, *default: str = ''*) → str

Attempt to guess the format of a path.

euporie.core.convert.mime.**get_mime**(*path:* Path | *str*) → str | None

Attempt to determine the mime-type of a path.

### euporie.core.convert.registry

Contain main format conversion function.

### Functions

| | |
|---|---|
| *NamedTuple*(typename[, fields]) | Typed version of namedtuple. |
| *find_route*(from_, to) | Find the shortest conversion path between two formats. |
| *register*(from_, to[, filter_, weight]) | Add a converter to the centralized format conversion system. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |

### euporie.core.convert.registry.NamedTuple

euporie.core.convert.registry.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)

Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

### euporie.core.convert.registry.find_route

euporie.core.convert.registry.**find_route**(*from_: str*, *to: str*) → list | None

Find the shortest conversion path between two formats.

### euporie.core.convert.registry.register

euporie.core.convert.registry.**register**(*from_: Iterable[str] | str*, *to: str*, *filter_: FilterOrBool = True*, *weight: int = 1*) → Callable

Add a converter to the centralized format conversion system.

### euporie.core.convert.registry.to_filter

euporie.core.convert.registry.**to_filter**(*bool_or_filter: Union[*Filter, *bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### Classes

| | |
|---|---|
| *Converter*(func, filter_[, weight]) | Hold a conversion function and its weight. |
| *FastDictCache*(get_value[, size]) | Fast, lightweight cache which keeps at most *size* items. |
| *SimpleCache*([maxsize]) | Very simple cache that discards the oldest item when the cache size is exceeded. |

### euporie.core.convert.registry.Converter

**class** euporie.core.convert.registry.**Converter**(*func: Callable*, *filter_:* Filter, *weight: int = 1*)

Hold a conversion function and its weight.

### euporie.core.convert.registry.FastDictCache

**class** euporie.core.convert.registry.**FastDictCache**(*get_value: Callable[[...], _V]*, *size: int = 1000000*)

Fast, lightweight cache which keeps at most *size* items. It will discard the oldest items in the cache first.

The cache is a dictionary, which doesn't keep track of access counts. It is perfect to cache little immutable objects which are not expensive to create, but where a dictionary lookup is still much faster than an object instantiation.

**Parameters**

**get_value** – Callable that's called in case of a missing key.

### euporie.core.convert.registry.SimpleCache

**class** euporie.core.convert.registry.**SimpleCache**(*maxsize: int = 8*)

Very simple cache that discards the oldest item when the cache size is exceeded.

**Parameters**

**maxsize** – Maximum size of the cache. (Don't make it too big.)

**class** euporie.core.convert.registry.**Converter**(*func: Callable*, *filter_: Filter*, *weight: int = 1*)

Bases: NamedTuple

Hold a conversion function and its weight.

**count**(*value*, */*)

Return number of occurrences of value.

**filter_: *Filter***

Alias for field number 1

**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 2

euporie.core.convert.registry.**find_route**(*from_: str*, *to: str*) → list | None

Find the shortest conversion path between two formats.

euporie.core.convert.registry.**register**(*from_: Iterable[str] | str*, *to: str*, *filter_: FilterOrBool = True*, *weight: int = 1*) → Callable

Add a converter to the centralized format conversion system.

### euporie.core.convert.utils

Utility functions for format converters.

## Functions

| | |
|---|---|
| *call_subproc*(data, cmd[, use_tempfile, suffix]) | Call the command as a subprocess and return it's output as bytes. |

### euporie.core.convert.utils.call_subproc

**async** euporie.core.convert.utils.**call_subproc**(*data: str | bytes*, *cmd: list[Any]*, *use_tempfile: bool = False*, *suffix: str = ''*) → 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
- **suffix** – Suffix for the temporary file name

**Returns**

The data printed to standard out by the subprocess.

## Classes

| | |
|---|---|
| *Path*(*args, **kwargs) | PurePath subclass that can make system calls. |

### euporie.core.convert.utils.Path

**class** euporie.core.convert.utils.**Path**(**args*, ***kwargs*)

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

**async** euporie.core.convert.utils.**call_subproc**(*data: str | bytes*, *cmd: list[Any]*, *use_tempfile: bool = False*, *suffix: str = ''*) → 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
- **suffix** – Suffix for the temporary file name

**Returns**

The data printed to standard out by the subprocess.

### euporie.core.current

Allow access to the current running application.

### Functions

| | |
|---|---|
| *get_app*() | Get the current active (running) Application. |

### euporie.core.current.get_app

euporie.core.current.**get_app**() → *BaseApp*

Get the current active (running) Application.

euporie.core.current.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.data_structures

Contain commonly used data structures.

### Functions

| | |
|---|---|
| *NamedTuple*(typename[, fields]) | Typed version of namedtuple. |
| *lru_cache*([maxsize, typed]) | Least-recently-used cache decorator. |

### euporie.core.data_structures.NamedTuple

euporie.core.data_structures.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)

Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

### euporie.core.data_structures.lru_cache

euporie.core.data_structures.**lru_cache**(*maxsize=128*, *typed=False*)

Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

## Classes

| | |
|---|---|
| *DiBool*([top, right, bottom, left]) | A tuple of four bools with directions. |
| *DiInt*([top, right, bottom, left]) | A tuple of four integers with directions. |
| *DiStr*([top, right, bottom, left]) | A tuple of four strings with directions. |
| *WeightedDiInt*(top, right, bottom, left) | A tuple of four weighted integers. |
| *WeightedInt*(weight, value) | Ainterger with an associated weight. |

### euporie.core.data_structures.DiBool

**class** euporie.core.data_structures.**DiBool**(*top: bool = False*, *right: bool = False*, *bottom: bool = False*, *left: bool = False*)

A tuple of four bools with directions.

### euporie.core.data_structures.DiInt

**class** euporie.core.data_structures.**DiInt**(*top: int = 0*, *right: int = 0*, *bottom: int = 0*, *left: int = 0*)

A tuple of four integers with directions.

### euporie.core.data_structures.DiStr

**class** euporie.core.data_structures.**DiStr**(*top: str = ''*, *right: str = ''*, *bottom: str = ''*, *left: str = ''*)

A tuple of four strings with directions.

### euporie.core.data_structures.WeightedDiInt

**class** euporie.core.data_structures.**WeightedDiInt**(*top:* WeightedInt, *right:* WeightedInt, *bottom:* WeightedInt, *left:* WeightedInt)

A tuple of four weighted integers.

### euporie.core.data_structures.WeightedInt

**class** euporie.core.data_structures.**WeightedInt**(*weight: int*, *value: int*)

Ainterger with an associated weight.

**class** euporie.core.data_structures.**DiBool**(*top: bool = False*, *right: bool = False*, *bottom: bool = False*, *left: bool = False*)

Bases: NamedTuple

A tuple of four bools with directions.

**bottom: bool**

Alias for field number 2

**count**(*value*, */*)

Return number of occurrences of value.

**classmethod from_value**(*value: bool*) → *DiBool*

Construct an instance from a single value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**left: bool**

Alias for field number 3

**right: bool**

Alias for field number 1

**top: bool**

Alias for field number 0

**class** euporie.core.data_structures.**DiInt**(*top: int = 0*, *right: int = 0*, *bottom: int = 0*, *left: int = 0*)

Bases: NamedTuple

A tuple of four integers with directions.

**bottom: int**

Alias for field number 2

**count**(*value*, */*)

Return number of occurrences of value.

**classmethod from_value**(*value: int*) → *DiInt*

Construct an instance from a single value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**left**: **int**

Alias for field number 3

**right**: **int**

Alias for field number 1

**top**: **int**

Alias for field number 0

**class** euporie.core.data_structures.**DiStr**(*top: str = ''*, *right: str = ''*, *bottom: str = ''*, *left: str = ''*)

Bases: NamedTuple

A tuple of four strings with directions.

**bottom**: **str**

Alias for field number 2

**count**(*value*, */*)

Return number of occurrences of value.

**classmethod from_value**(*value: str*) → *DiStr*

Construct an instance from a single value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**left**: **str**

Alias for field number 3

**right**: **str**

Alias for field number 1

**top**: **str**

Alias for field number 0

**class** euporie.core.data_structures.**WeightedDiInt**(*top:* WeightedInt, *right:* WeightedInt, *bottom:* WeightedInt, *left:* WeightedInt)

Bases: NamedTuple

A tuple of four weighted integers.

**bottom**: ***WeightedInt***

Alias for field number 2

**count**(*value*, */*)

Return number of occurrences of value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**left**: *WeightedInt*
Alias for field number 3

**right**: *WeightedInt*
Alias for field number 1

**top**: *WeightedInt*
Alias for field number 0

**property unweighted**: *DiInt*
Get the padding without weights.

**class** euporie.core.data_structures.**WeightedInt**(*weight: int*, *value: int*)
Bases: NamedTuple

Ainterger with an associated weight.

**count**(*value*, */*)
Return number of occurrences of value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)
Return first index of value.

Raises ValueError if the value is not present.

**value**: **int**
Alias for field number 1

**weight**: **int**
Alias for field number 0

### euporie.core.diagnostics

Contains container classes for diagnostic information.

### Functions

| | |
|---|---|
| *NamedTuple*(typename[, fields]) | Typed version of namedtuple. |

### euporie.core.diagnostics.NamedTuple

euporie.core.diagnostics.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)
Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

## Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *Diagnostic*(code, message, level, link, ...) | Represent a diagnostic item for code. |
| *Report*([iterable]) | Class for storing a diagnostic report. |

### euporie.core.diagnostics.ABCMeta

**class** euporie.core.diagnostics.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.diagnostics.Diagnostic

**class** euporie.core.diagnostics.**Diagnostic**(*code: str*, *message: str*, *level: Literal[0, 1, 2, 3, 4, 5]*, *link: str*, *lines: slice*, *chars: slice*)

Represent a diagnostic item for code.

### euporie.core.diagnostics.Report

**class** euporie.core.diagnostics.**Report**(*iterable=()*, */*)

Class for storing a diagnostic report.

**class** euporie.core.diagnostics.**Diagnostic**(*code: str*, *message: str*, *level: Literal[0, 1, 2, 3, 4, 5]*, *link: str*, *lines: slice*, *chars: slice*)

Bases: NamedTuple

Represent a diagnostic item for code.

**chars: slice**

Alias for field number 5

**code: str**

Alias for field number 0

**count**(*value*, */*)

Return number of occurrences of value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**level: Literal[0, 1, 2, 3, 4, 5]**

Alias for field number 2

**lines: slice**

Alias for field number 4

**link: str**

Alias for field number 3

**message: str**

Alias for field number 1

**class** euporie.core.diagnostics.**Report**(*iterable=()*, */*)

Bases: List[*Diagnostic*]

Class for storing a diagnostic report.

**append**(*object*, */*)

Append object to the end of the list.

**clear**()

Remove all items from list.

**copy**()

Return a shallow copy of the list.

**count**(*value*, */*)

Return number of occurrences of value.

**extend**(*iterable*, */*)

Extend list by appending elements from the iterable.

**classmethod from_lsp**(*text: str*, *outputs: list[dict]*) → *Report*

Create a diagnostic report from LSP output.

**classmethod from_reports**(**reports:* Report) → *Report*

Initialize a new report.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**insert**(*index*, *object*, */*)

Insert object before index.

**pop**(*index=-1*, */*)

Remove and return item at index (default last).

Raises IndexError if list is empty or index is out of range.

**remove**(*value*, */*)

Remove first occurrence of value.

Raises ValueError if the value is not present.

**reverse**()

Reverse *IN PLACE*.

**sort**(*, *key=None*, *reverse=False*)

Sort the list in ascending order and return None.

The sort is in-place (i.e. the list itself is modified) and stable (i.e. the order of two equal elements is maintained).

If a key function is given, apply it once to each list item and sort them, ascending or descending, according to their function values.

The reverse flag can be set to sort in descending order.

## euporie.core.filters

Define common filters.

### Module Attributes

| | |
|---|---|
| *insert_mode* | Determine if any binding style is in replace mode. |

## euporie.core.filters.insert_mode

euporie.core.filters.**insert_mode = Condition(<function vi_mode>)&Condition(<function vi_insert_mode>)|Condition(<function emacs_mode>)&Condition(<function emacs_insert_mode>)|Condition(<function micro_mode>)&Condition(<function micro_insert_mode>)**

Determine if any binding style is in replace mode.

### Functions

| | |
|---|---|
| *command_exists*(*cmds) | Verify a list of external commands exist on the system. |
| *have_modules*(*modules) | Verify a list of python modules are importable. |
| *import_module*(name[, package]) | Import a module. |
| *reduce*(function, iterable[, initial]) | Apply a function of two arguments cumulatively to the items of a sequence or iterable, from left to right, so as to reduce the iterable to a single value. |
| *scrollable*(window) | Return a filter which indicates if a window is scrollable. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |
| *which*(cmd[, mode, path]) | Given a command, mode, and a PATH string, return the path which conforms to the given mode on the PATH, or None if there is no such file. |

### euporie.core.filters.command_exists

euporie.core.filters.**command_exists**(**cmds: str*) → *Filter*

Verify a list of external commands exist on the system.

### euporie.core.filters.have_modules

euporie.core.filters.**have_modules**(**modules: str*) → *Filter*

Verify a list of python modules are importable.

### euporie.core.filters.import_module

euporie.core.filters.**import_module**(*name*, *package=None*)

Import a module.

The 'package' argument is required when performing a relative import. It specifies the package to use as the anchor point from which to resolve the relative import to an absolute import.

### euporie.core.filters.reduce

euporie.core.filters.**reduce**(*function*, *iterable*[, *initial* ]) → value

Apply a function of two arguments cumulatively to the items of a sequence or iterable, from left to right, so as to reduce the iterable to a single value. For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates ((((1+2)+3)+4)+5). If initial is present, it is placed before the items of the iterable in the calculation, and serves as a default when the iterable is empty.

### euporie.core.filters.scrollable

euporie.core.filters.**scrollable**(*window:* Window) → *Filter*

Return a filter which indicates if a window is scrollable.

### euporie.core.filters.to_filter

euporie.core.filters.**to_filter**(*bool_or_filter: Union[*Filter*, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### euporie.core.filters.which

euporie.core.filters.**which**(*cmd*, *mode=1*, *path=None*)

Given a command, mode, and a PATH string, return the path which conforms to the given mode on the PATH, or None if there is no such file.

*mode* defaults to os.F_OK | os.X_OK. *path* defaults to the result of os.environ.get("PATH"), or can be overridden with a custom search path.

## Classes

| | |
|---|---|
| *Condition*(func) | Turn any callable into a Filter. |
| *EditingMode*(value[, names, module, ...]) | |
| *MicroInputMode*(value[, names, module, ...]) | Enum to define edit mode state types. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.filters.Condition

**class** euporie.core.filters.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.filters.EditingMode

**class** euporie.core.filters.**EditingMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.filters.MicroInputMode

**class** euporie.core.filters.**MicroInputMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Enum to define edit mode state types.

### euporie.core.filters.partial

**class** euporie.core.filters.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

euporie.core.filters.**command_exists**(**cmds: str*) → *Filter*

Verify a list of external commands exist on the system.

euporie.core.filters.**have_modules**(**modules: str*) → *Filter*

Verify a list of python modules are importable.

euporie.core.filters.**insert_mode = Condition(<function vi_mode>)&Condition(<function vi_insert_mode>)|Condition(<function emacs_mode>)&Condition(<function emacs_insert_mode>)|Condition(<function micro_mode>)&Condition(<function micro_insert_mode>)**

Determine if any binding style is in replace mode.

euporie.core.filters.**scrollable**(*window:* Window) → *Filter*

Return a filter which indicates if a window is scrollable.

### euporie.core.format

Contain functions to automatically format code cell input.

### Functions

| | |
|---|---|
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |
| *command_exists*(*cmds) | Verify a list of external commands exist on the system. |
| *range_to_slice*(start_line, start_char, ...) | Convert a line/character ranger to a slice. |

### euporie.core.format.abstractmethod

euporie.core.format.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):

…

### euporie.core.format.command_exists

euporie.core.format.**command_exists**(**cmds: str*) → *Filter*

Verify a list of external commands exist on the system.

### euporie.core.format.range_to_slice

euporie.core.format.**range_to_slice**(*start_line: int*, *start_char: int*, *end_line: int*, *end_char: int*, *text: str*) → slice

Convert a line/character ranger to a slice.

### Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *CliFormatter*(command[, languages]) | Format using an external command. |
| *Formatter*([languages]) | Text formatter class which reformats text. |
| *LspFormatter*(lsp, path[, languages]) | Format a document using a LSP server. |

#### euporie.core.format.ABCMeta

**class** euporie.core.format.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

#### euporie.core.format.CliFormatter

**class** euporie.core.format.**CliFormatter**(*command: list[str]*, *languages: set[str] | None = None*)

Format using an external command.

#### euporie.core.format.Formatter

**class** euporie.core.format.**Formatter**(*languages: set[str] | None = None*)

Text formatter class which reformats text.

#### euporie.core.format.LspFormatter

**class** euporie.core.format.**LspFormatter**(*lsp: LspClient*, *path: Path*, *languages: set[str] | None = None*)

Format a document using a LSP server.

**class** euporie.core.format.**CliFormatter**(*command: list[str]*, *languages: set[str] | None = None*)

Bases: *Formatter*

Format using an external command.

**format**(*text: str*) → str

Pass the text to the command over stdin and return the output.

**class** euporie.core.format.**Formatter**(*languages: set[str] | None = None*)

Bases: object

Text formatter class which reformats text.

**abstract format**(*text: str*) → str

Format the string.

**class** euporie.core.format.**LspFormatter**(*lsp:* LspClient, *path:* Path, *languages: set[str]* | *None* = *None*)

Bases: *Formatter*

Format a document using a LSP server.

**format**(*text: str*) → str

Format the url with the LSP.

The LSP already knows the contents of the file, which is why we do not pass the text to the formatting function.

### euporie.core.ft

Contain modules for working with formatted text.

### Modules

| | |
|---|---|
| *euporie.core.ft.ansi* | Contain ANSI formatted text parser. |
| *euporie.core.ft.html* | Contain a HTML to formatted text parser. |
| *euporie.core.ft.table* | Allow drawing tables as `FormattedText`. |
| *euporie.core.ft.utils* | Utilities for manipulating formatted text. |

### euporie.core.ft.ansi

Contain ANSI formatted text parser.

### Classes

| | |
|---|---|
| *ANSI*(value[, tab_size]) | Convert ANSI text into formatted text, preserving all control sequences. |
| *PTANSI* | alias of `ANSI` |

### euporie.core.ft.ansi.ANSI

**class** euporie.core.ft.ansi.**ANSI**(*value: str*, *tab_size: int* = *8*)

Convert ANSI text into formatted text, preserving all control sequences.

### euporie.core.ft.ansi.PTANSI

euporie.core.ft.ansi.**PTANSI**

alias of `ANSI`

**class** euporie.core.ft.ansi.**ANSI**(*value: str*, *tab_size: int* = *8*)

Bases: `ANSI`

Convert ANSI text into formatted text, preserving all control sequences.

**format** (**args: str*, ***kwargs: str*) → ANSI

Like *str.format*, but make sure that the arguments are properly escaped. (No ANSI escapes can be injected.)

### euporie.core.ft.html

Contain a HTML to formatted text parser.

### Functions

| | |
|---|---|
| NamedTuple(typename[, fields]) | Typed version of namedtuple. |
| add_border(ft[, width, style, border_grid, ...]) | Add a border around formatted text. |
| align(ft[, how, width, style, placeholder, ...]) | Align formatted text at a given width. |
| apply_reverse_overwrites(ft) | Write fragments tagged with "[ReverseOverwrite]" over text to their left. |
| apply_style(ft, style) | Apply a style to formatted text. |
| bisect_right(a, x[, lo, hi, key]) | Return the index where to insert item x in list a, assuming a is sorted. |
| cast(typ, val) | Cast a value to a type. |
| ceil(x, /) | Return the ceiling of x as an Integral. |
| compute_padding(cell[, render_count]) | Compute a cell's padding. |
| concat(ft_a, ft_b[, baseline_a, baseline_b, ...]) | Concatenate two blocks of formatted text, aligning at a given baseline. |
| css_dimension(value[, vertical, available]) | Convert CSS dimensions to terminal cell sizes. |
| eq(a, b, /) | Same as a == b. |
| fragment_list_to_words(fragments[, sep]) | Split formatted text into a list of word fragments which form words. |
| fragment_list_width(fragments) | Return the character width of this text fragment list. |
| ge(a, b, /) | Same as a >= b. |
| get_app() | Get the current active (running) Application. |
| get_app_session() | |
| get_color(value) | Extract a hex color from a string. |
| get_format(path[, default]) | Attempt to guess the format of a path. |
| get_integer(value) | Extract the first integer from a string. |
| get_loop() | Create or return the conversion IO loop. |
| gt(a, b, /) | Same as a > b. |
| join_lines(fragments) | Join a list of lines of formatted text. |
| last_char(ft) | Retrieve the last character of formatted text. |
| le(a, b, /) | Same as a <= b. |
| literal_eval(node_or_string) | Evaluate an expression node or a string containing only a Python expression. |
| lru_cache([maxsize, typed]) | Least-recently-used cache decorator. |
| lt(a, b, /) | Same as a < b. |
| match_css_selector(selector, attrs, pseudo, ...) | Determine if a CSS selector matches a particular element. |
| max_line_width(ft) | Calculate the length of the longest line in formatted text. |
| overload(func) | Decorator for overloaded functions/methods. |
| pad(ft[, width, char, style]) | Fill space at the end of lines. |
| parse_css_content(content) | Convert CSS declarations into the internals style representation. |
| parse_media_condition(condition, dom) | Convert media rules to conditions. |

continues on next page

Table 6 – continued from previous page

| | |
|---|---|
| *parse_style_sheet*(css_str, dom[, condition]) | Collect all CSS styles from style tags. |
| *paste*(ft_top, ft_bottom[, row, col, transparent]) | Pate formatted text on top of other formatted text. |
| *selector_specificity*(selector_parts) | Calculate the specificity score of a CSS selector. |
| *split_lines*(fragments) | Take a single list of (style_str, text) tuples and yield one such list for each line. |
| *strip*(ft[, left, right, chars, only_unstyled]) | Strip whitespace (or a given character) from the ends of formatted text. |
| *truncate*(ft, width[, style, placeholder, ...]) | Truncate all lines at a given length. |
| *try_eval*(value[, default]) | Attempt to cast a string to a python type. |
| *url_to_fs*(url, **kwargs) | Turn fully-qualified and potentially chained URL into filesystem instance |
| *valign*(ft[, how, height, style]) | Align formatted text vertically. |

### euporie.core.ft.html.NamedTuple

euporie.core.ft.html.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)

Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

### euporie.core.ft.html.add_border

euporie.core.ft.html.**add_border**(*ft: StyleAndTextTuples*, *width: int | None = None*, *style: str = ''*, *border_grid: GridStyle = □□□□ □ □□ □□□□ □□□□*, *border_visibility: DiBool | bool = True*, *border_style: DiStr | str = ''*, *padding: DiInt | int = 0*, *padding_style: DiStr | str = ''*) → StyleAndTextTuples

Add a border around formatted text.

**Parameters**

- **ft** – The formatted text to enclose with a border
- **width** – The target width including the border and padding
- **style** – The style to apply to the content background
- **border_grid** – The grid style to use for the border
- **border_visibility** – Determines which edges should receive a border

- **border_style** – The style to apply to the border
- **padding** – The width of spacing to apply between the content and the border
- **padding_style** – The style to apply to the border

**Returns**
The indented formatted text

### euporie.core.ft.html.align

euporie.core.ft.html.**align**(*ft: StyleAndTextTuples*, *how:* FormattedTextAlign = *FormattedTextAlign.LEFT*, *width: int | None = None*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Align formatted text at a given width.

**Parameters**

- **how** – The alignment direction
- **ft** – The formatted text to strip
- **width** – The width to which the output should be padded. If `None`, the length of the longest line is used
- **style** – The style to apply to the padding
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – If True, whitespace will be ignored

**Returns**
The aligned formatted text

### euporie.core.ft.html.apply_reverse_overwrites

euporie.core.ft.html.**apply_reverse_overwrites**(*ft: StyleAndTextTuples*) → StyleAndTextTuples

Write fragments tagged with "[ReverseOverwrite]" over text to their left.

### euporie.core.ft.html.apply_style

euporie.core.ft.html.**apply_style**(*ft: StyleAndTextTuples*, *style: str*) → StyleAndTextTuples

Apply a style to formatted text.

### euporie.core.ft.html.bisect_right

euporie.core.ft.html.**bisect_right**(*a*, *x*, *lo=0*, *hi=None*, ***, *key=None*)

Return the index where to insert item x in list a, assuming a is sorted.

The return value i is such that all e in a[:i] have e <= x, and all e in a[i:] have e > x. So if x already appears in the list, a.insert(i, x) will insert just after the rightmost x already there.

Optional args lo (default 0) and hi (default len(a)) bound the slice of a to be searched.

A custom key function can be supplied to customize the sort order.

### euporie.core.ft.html.cast

euporie.core.ft.html.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.ft.html.ceil

euporie.core.ft.html.**ceil**(*x*, */*)

Return the ceiling of x as an Integral.

This is the smallest integer >= x.

### euporie.core.ft.html.compute_padding

euporie.core.ft.html.**compute_padding**(*cell:* Cell, *render_count:* *int* = *0*) → *DiInt*

Compute a cell's padding.

### euporie.core.ft.html.concat

euporie.core.ft.html.**concat**(*ft_a: StyleAndTextTuples*, *ft_b: StyleAndTextTuples*, *baseline_a:* *int* = *0*, *baseline_b:* *int* = *0*, *style:* *str* = *''*) → tuple[StyleAndTextTuples, int]

Concatenate two blocks of formatted text, aligning at a given baseline.

**Parameters**

- **ft_a** – The first block of formatted text to combine
- **ft_b** – The second block of formatted text to combine
- **baseline_a** – The row to use to align the first block of formatted text with the second, counted in lines down from the top of the block
- **baseline_b** – The row to use to align the second block of formatted text with the second, counted in lines down from the top of the block
- **style** – The style to use for any extra lines added

**Returns**

A tuple containing the combined formatted text and the new baseline position

### euporie.core.ft.html.css_dimension

euporie.core.ft.html.**css_dimension**(*value:* *str*, *vertical:* *bool* = *False*, *available:* *float* | *int* | *None* = *None*) → float | None

Convert CSS dimensions to terminal cell sizes.

### euporie.core.ft.html.eq

euporie.core.ft.html.**eq**(*a*, *b*, */*)

Same as a == b.

### euporie.core.ft.html.fragment_list_to_words

euporie.core.ft.html.**fragment_list_to_words**(*fragments: StyleAndTextTuples*, *sep: str = ' '*) → Iterable[StyleAndTextTuples]

Split formatted text into a list of word fragments which form words.

### euporie.core.ft.html.fragment_list_width

euporie.core.ft.html.**fragment_list_width**(*fragments: StyleAndTextTuples*) → int

Return the character width of this text fragment list.

Takes double width characters into account, and ignore special fragments: * ZeroWidthEscape * ReverseOverwrite

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

**Returns**

The width of the fragment list

### euporie.core.ft.html.ge

euporie.core.ft.html.**ge**(*a*, *b*, */*)

Same as a >= b.

### euporie.core.ft.html.get_app

euporie.core.ft.html.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.ft.html.get_app_session

euporie.core.ft.html.**get_app_session**() → AppSession

### euporie.core.ft.html.get_color

euporie.core.ft.html.**get_color**(*value: str*) → str

Extract a hex color from a string.

### euporie.core.ft.html.get_format

euporie.core.ft.html.**get_format**(*path:* Path | *str*, *default: str* = '') → str

Attempt to guess the format of a path.

### euporie.core.ft.html.get_integer

euporie.core.ft.html.**get_integer**(*value: str*) → int | None

Extract the first integer from a string.

### euporie.core.ft.html.get_loop

euporie.core.ft.html.**get_loop**() → AbstractEventLoop

Create or return the conversion IO loop.

The loop will be running on a separate thread.

### euporie.core.ft.html.gt

euporie.core.ft.html.**gt**(*a*, *b*, */*)

Same as a > b.

### euporie.core.ft.html.join_lines

euporie.core.ft.html.**join_lines**(*fragments: list[StyleAndTextTuples]*) → StyleAndTextTuples

Join a list of lines of formatted text.

### euporie.core.ft.html.last_char

euporie.core.ft.html.**last_char**(*ft: StyleAndTextTuples*) → str | None

Retrieve the last character of formatted text.

### euporie.core.ft.html.le

euporie.core.ft.html.**le**(*a*, *b*, */*)

Same as a <= b.

### euporie.core.ft.html.literal_eval

euporie.core.ft.html.**literal_eval**(*node_or_string*)

Evaluate an expression node or a string containing only a Python expression. The string or node provided may only consist of the following Python literal structures: strings, bytes, numbers, tuples, lists, dicts, sets, booleans, and None.

Caution: A complex expression can overflow the C stack and cause a crash.

### euporie.core.ft.html.lru_cache

euporie.core.ft.html.**lru_cache**(*maxsize=128*, *typed=False*)

Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

### euporie.core.ft.html.lt

euporie.core.ft.html.**lt**(*a*, *b*, */*)

Same as a < b.

### euporie.core.ft.html.match_css_selector

euporie.core.ft.html.**match_css_selector**(*selector: str*, *attrs: str*, *pseudo: str*, *element_name: str*, *is_first_child_element: bool*, *is_last_child_element: bool*, *sibling_element_index: int | None*, ***element_attrs: Any*) → bool

Determine if a CSS selector matches a particular element.

### euporie.core.ft.html.max_line_width

euporie.core.ft.html.**max_line_width**(*ft: StyleAndTextTuples*) → int

Calculate the length of the longest line in formatted text.

### euporie.core.ft.html.overload

euporie.core.ft.html.**overload**(*func*)

Decorator for overloaded functions/methods.

In a stub file, place two or more stub definitions for the same function in a row, each decorated with @overload.

For example:

```
@overload
def utf8(value: None) -> None: ...
@overload
def utf8(value: bytes) -> bytes: ...
@overload
def utf8(value: str) -> bytes: ...
```

In a non-stub file (i.e. a regular .py file), do the same but follow it with an implementation. The implementation should *not* be decorated with @overload:

```
@overload
def utf8(value: None) -> None: ...
@overload
def utf8(value: bytes) -> bytes: ...
@overload
def utf8(value: str) -> bytes: ...
def utf8(value):
    ...  # implementation goes here
```

The overloads for a function can be retrieved at runtime using the get_overloads() function.

### euporie.core.ft.html.pad

euporie.core.ft.html.**pad**(*ft: StyleAndTextTuples*, *width: int | None = None*, *char: str = ' '*, *style: str = ''*) → StyleAndTextTuples

Fill space at the end of lines.

### euporie.core.ft.html.parse_css_content

euporie.core.ft.html.**parse_css_content**(*content: str*) → dict[str, str]

Convert CSS declarations into the internals style representation.

### euporie.core.ft.html.parse_media_condition

euporie.core.ft.html.**parse_media_condition**(*condition: str*, *dom:* HTML) → *Filter*

Convert media rules to conditions.

### euporie.core.ft.html.parse_style_sheet

euporie.core.ft.html.**parse_style_sheet**(*css_str: str*, *dom: HTML*, *condition: Filter = <prompt_toolkit.filters.base.Always object>*) → None

Collect all CSS styles from style tags.

### euporie.core.ft.html.paste

euporie.core.ft.html.**paste**(*ft_top: StyleAndTextTuples*, *ft_bottom: StyleAndTextTuples*, *row: int = 0*, *col: int = 0*, *transparent: bool = False*) → StyleAndTextTuples

Pate formatted text on top of other formatted text.

### euporie.core.ft.html.selector_specificity

euporie.core.ft.html.**selector_specificity**(*selector_parts: tuple[*euporie.core.ft.html.CssSelector, *...]*) → tuple[int, int, int]

Calculate the specificity score of a CSS selector.

### euporie.core.ft.html.split_lines

euporie.core.ft.html.**split_lines**(*fragments: Iterable[OneStyleAndTextTuple]*) → Iterable[StyleAndTextTuples]

Take a single list of (style_str, text) tuples and yield one such list for each line. Just like str.split, this will yield at least one item.

**Parameters**

**fragments** – Iterable of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.ft.html.strip

euporie.core.ft.html.**strip**(*ft: StyleAndTextTuples*, *left: bool = True*, *right: bool = True*, *chars: str | None = None*, *only_unstyled: bool = False*) → StyleAndTextTuples

Strip whitespace (or a given character) from the ends of formatted text.

**Parameters**

- **ft** – The formatted text to strip
- **left** – If `True`, strip from the left side of the input
- **right** – If `True`, strip from the right side of the input
- **chars** – The character to strip. If `None`, strips whitespace

- **only_unstyled** – If `True`, only strip unstyled fragments

**Returns**
The stripped formatted text

### euporie.core.ft.html.truncate

euporie.core.ft.html.**truncate**(*ft: StyleAndTextTuples*, *width: int*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Truncate all lines at a given length.

**Parameters**

- **ft** – The formatted text to truncate
- **width** – The width at which to truncate the text
- **style** – The style to apply to the truncation placeholder. The style of the truncated text will be used if not provided
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – Do not use placeholder when truncating whitespace

**Returns**
The truncated formatted text

### euporie.core.ft.html.try_eval

euporie.core.ft.html.**try_eval**(*value: str*, *default: Any = None*) → Any

Attempt to cast a string to a python type.

### euporie.core.ft.html.url_to_fs

euporie.core.ft.html.**url_to_fs**(*url*, ***kwargs*)

Turn fully-qualified and potentially chained URL into filesystem instance

**Parameters**

- **url** (*str*) – The fsspec-compatible URL
- ****kwargs** (*dict*) – Extra options that make sense to a particular storage connection, e.g. host, port, username, password, etc.

**Returns**

- **filesystem** (*FileSystem*) – The new filesystem discovered from `url` and created with `**kwargs`.
- **urlpath** (*str*) – The file-systems-specific URL for `url`.

### euporie.core.ft.html.valign

euporie.core.ft.html.**valign**(*ft: StyleAndTextTuples*, *how:* FormattedTextVerticalAlign = *FormattedTextVerticalAlign.MIDDLE*, *height: int | None = None*, *style: str = ''*) → StyleAndTextTuples

Align formatted text vertically.

### Classes

| | |
|---|---|
| *Cell*([text, row, col, colspan, rowspan, ...]) | A table cell. |
| *Condition*(func) | Turn any callable into a Filter. |
| *CssSelector*([comb, item, attr, pseudo]) | A named tuple to hold CSS selector data. |
| *CustomHTMLParser*(dom) | An HTML parser. |
| *Datum*(data, *args, **kwargs) | Class for storing and converting display data. |
| *DiBool*([top, right, bottom, left]) | A tuple of four bools with directions. |
| *DiInt*([top, right, bottom, left]) | A tuple of four integers with directions. |
| *DiLineStyle*([top, right, bottom, left]) | A description of a cell border: a `LineStyle` for each edge. |
| *DiStr*([top, right, bottom, left]) | A tuple of four strings with directions. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *Direction*([x, y]) | A description of a direction. |
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *FormattedTextAlign*(value[, names, module, ...]) | Alignment of formatted text. |
| *FormattedTextVerticalAlign*(value[, names, ...]) | Vertical alignment of formatted text. |
| *GridStyle*(line_style, mask) | A collection of characters which can be used to draw a grid. |
| *HTML*(markup[, base, width, height, ...]) | A HTML formatted text renderer. |
| *HTMLParser*(*[, convert_charrefs]) | Find tags and other markup and call handler functions. |
| *Mapping*() | A Mapping is a generic container for associating key/value pairs. |
| *Node*(dom, name, parent[, text, attrs, contents]) | Represent an node in the DOM. |
| *Size*(rows, columns) | |
| *Table*([rows, cols, width, expand, align, ...]) | A table. |
| *Theme*(element, parent_theme[, ...]) | The computed theme of an element. |
| *UPath*(*args[, protocol]) | |
| *WindowAlign*(value[, names, module, ...]) | Alignment of the Window content. |
| *cached_property*(func) | |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |
| *zip_longest* | zip_longest(iter1 [,iter2 [...]], [fillvalue=None]) --> zip_longest object |

### euporie.core.ft.html.Cell

**class** euporie.core.ft.html.**Cell**(*text: AnyFormattedText = ''*, *row: Row | None = None*, *col: Col | None = None*, *colspan: int = 1*, *rowspan: int = 1*, *width: int | None = None*, *align: FormattedTextAlign | None = None*, *style: str = ''*, *padding: DiInt | int = 0*, *border_line: DiLineStyle | LineStyle = LineStyle(Thin)*, *border_style: DiStr | str = ''*, *border_visibility: DiBool | bool | None = True*)

A table cell.

### euporie.core.ft.html.Condition

**class** euporie.core.ft.html.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.ft.html.CssSelector

**class** euporie.core.ft.html.**CssSelector**(*comb: str | None = None*, *item: str | None = None*, *attr: str | None = None*, *pseudo: str | None = None*)

A named tuple to hold CSS selector data.

### euporie.core.ft.html.CustomHTMLParser

**class** euporie.core.ft.html.**CustomHTMLParser**(*dom: HTML*)

An HTML parser.

### euporie.core.ft.html.Datum

**class** euporie.core.ft.html.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Class for storing and converting display data.

### euporie.core.ft.html.DiBool

**class** euporie.core.ft.html.**DiBool**(*top: bool = False*, *right: bool = False*, *bottom: bool = False*, *left: bool = False*)

A tuple of four bools with directions.

### euporie.core.ft.html.DiInt

**class** euporie.core.ft.html.**DiInt**(*top: int = 0*, *right: int = 0*, *bottom: int = 0*, *left: int = 0*)

A tuple of four integers with directions.

### euporie.core.ft.html.DiLineStyle

**class** euporie.core.ft.html.**DiLineStyle**(*top:* LineStyle *= LineStyle(None)*, *right:* LineStyle *= LineStyle(None)*, *bottom:* LineStyle *= LineStyle(None)*, *left:* LineStyle *= LineStyle(None)*)

A description of a cell border: a `LineStyle` for each edge.

### euporie.core.ft.html.DiStr

**class** euporie.core.ft.html.**DiStr**(*top: str = ''*, *right: str = ''*, *bottom: str = ''*, *left: str = ''*)

A tuple of four strings with directions.

### euporie.core.ft.html.Dimension

**class** euporie.core.ft.html.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.ft.html.Direction

**class** euporie.core.ft.html.**Direction**(*x: bool = False*, *y: bool = False*)

A description of a direction.

### euporie.core.ft.html.Event

**class** euporie.core.ft.html.**Event**(*sender: _Sender*, *handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.ft.html.FormattedTextAlign

**class** euporie.core.ft.html.**FormattedTextAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of formatted text.

### euporie.core.ft.html.FormattedTextVerticalAlign

**class** euporie.core.ft.html.**FormattedTextVerticalAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Vertical alignment of formatted text.

### euporie.core.ft.html.GridStyle

**class** euporie.core.ft.html.**GridStyle**(*line_style: ~euporie.core.border.LineStyle = LineStyle(None)*, *mask: ~euporie.core.border.Mask = <euporie.core.border.Mask object>*)

A collection of characters which can be used to draw a grid.

### euporie.core.ft.html.HTML

**class** euporie.core.ft.html.**HTML**(*markup: str*, *base: Path | str | None = None*, *width: int | None = None*, *height: int | None = None*, *collapse_root_margin: bool = False*, *fill: bool = True*, *css: CssSelectors | None = None*, *browser_css: CssSelectors | None = None*, *mouse_handler: Callable[[Node, MouseEvent], NotImplementedOrNone] | None = None*, *paste_fixed: bool = True*, *defer_assets: bool = False*, *on_update: Callable[[HTML], None] | None = None*, *on_change: Callable[[HTML], None] | None = None*, *_initial_format: str = ''*)

A HTML formatted text renderer.

Accepts a HTML string and renders it at a given width.

### euporie.core.ft.html.HTMLParser

**class** euporie.core.ft.html.**HTMLParser**(*, *convert_charrefs=True*)

Find tags and other markup and call handler functions.

**Usage:**

p = HTMLParser() p.feed(data) … p.close()

Start tags are handled by calling self.handle_starttag() or self.handle_startendtag(); end tags by self.handle_endtag(). The data between tags is passed from the parser to the derived class by calling self.handle_data() with the data as argument (the data may be split up in arbitrary chunks). If convert_charrefs is True the character references are converted automatically to the corresponding Unicode character (and self.handle_data() is no longer split in chunks), otherwise they are passed by calling self.handle_entityref() or self.handle_charref() with the string containing respectively the named or numeric reference as the argument.

### euporie.core.ft.html.Mapping

**class** euporie.core.ft.html.**Mapping**

A Mapping is a generic container for associating key/value pairs.

This class provides concrete generic implementations of all methods except for __getitem__, __iter__, and __len__.

### euporie.core.ft.html.Node

**class** euporie.core.ft.html.**Node**(*dom:* HTML, *name: str*, *parent:* euporie.core.ft.html.Node | *None*, *text: str* = '', *attrs: list[tuple[str, str | None]] | None* = None, *contents: list[*euporie.core.ft.html.Node*] | None* = None)

Represent an node in the DOM.

### euporie.core.ft.html.Size

**class** euporie.core.ft.html.**Size**(*rows*, *columns*)

### euporie.core.ft.html.Table

**class** euporie.core.ft.html.**Table**(*rows: Sequence[*Row*] | None* = None, *cols: Sequence[*Col*] | None* = None, *width: AnyDimension | None* = None, *expand: bool* = False, *align:* FormattedTextAlign = *FormattedTextAlign.LEFT*, *style: str* = '', *padding:* DiInt | *int* | *None* = None, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = '', *border_visibility:* DiBool | *bool* = False, *background_style: str* = '')

A table.

### euporie.core.ft.html.Theme

**class** euporie.core.ft.html.**Theme**(*element:* Node, *parent_theme:* euporie.core.ft.html.Theme | *None*, *available_width: int* = 0, *available_height: int* = 0)

The computed theme of an element.

### euporie.core.ft.html.UPath

**class** euporie.core.ft.html.**UPath**(**args*, *protocol: str | None* = None, ***storage_options: Any*)

### euporie.core.ft.html.WindowAlign

**class** euporie.core.ft.html.**WindowAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of the Window content.

Note that this is different from *HorizontalAlign* and *VerticalAlign*, which are used for the alignment of the child containers in respectively *VSplit* and *HSplit*.

### euporie.core.ft.html.cached_property

**class** euporie.core.ft.html.**cached_property**(*func*)

### euporie.core.ft.html.partial

**class** euporie.core.ft.html.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

### euporie.core.ft.html.zip_longest

**class** euporie.core.ft.html.**zip_longest**

zip_longest(iter1 [,iter2 […]], [fillvalue=None]) –> zip_longest object

Return a zip_longest object whose .__next__() method returns a tuple where the i-th element comes from the i-th iterable argument. The .__next__() method continues until the longest iterable in the argument sequence is exhausted and then it raises StopIteration. When the shorter iterables are exhausted, the fillvalue is substituted in their place. The fillvalue defaults to None or can be specified by a keyword argument.

**class** euporie.core.ft.html.**CssSelector**(*comb: str | None = None*, *item: str | None = None*, *attr: str | None = None*, *pseudo: str | None = None*)

Bases: NamedTuple

A named tuple to hold CSS selector data.

**attr: str | None**

Alias for field number 2

**comb: str | None**

Alias for field number 0

**count**(*value*, */*)

Return number of occurrences of value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**item: str | None**

Alias for field number 1

**pseudo: str | None**

Alias for field number 3

**class** euporie.core.ft.html.**CustomHTMLParser**(*dom:* HTML)

Bases: *HTMLParser*

An HTML parser.

**CDATA_CONTENT_ELEMENTS = ('script', 'style')**

**autoclose**() → None

Automatically close void elements.

**check_for_whole_start_tag**(*i*)

**clear_cdata_mode**()

**close**()
Handle any buffered data.

**curr**: *Node*

**feed**(*data*)
Feed data to the parser.

Call this as often as you want, with as little or as much text as you want (may include ‘n’).

**get_starttag_text**()
Return full source of start tag: ‘<…>’.

**getpos**()
Return current line number and offset.

**goahead**(*end*)

**handle_charref**(*name*)

**handle_comment**(*data*)

**handle_data**(*data: str*) → None
Create data (text) elements.

**handle_decl**(*decl*)

**handle_endtag**(*tag: str*) → None
Handle end tags: close the currently opened element.

**handle_entityref**(*name*)

**handle_pi**(*data*)

**handle_startendtag**(*tag*, *attrs*)

**handle_starttag**(*tag: str*, *attrs: list[tuple[str, str | None]]*) → None
Open a new element.

**parse**(*markup: str*) → *Node*
Pare HTML markup.

**parse_bogus_comment**(*i*, *report=1*)

**parse_comment**(*i*, *report=1*)

**parse_declaration**(*i*)

**parse_endtag**(*i*)

**parse_html_declaration**(*i*)

**parse_marked_section**(*i*, *report=1*)

**parse_pi**(*i*)

**parse_starttag**(*i*)

**reset**()

Reset this instance. Loses all unprocessed data.

**set_cdata_mode**(*elem*)

**soup: *Node***

**unknown_decl**(*data*)

**updatepos**(*i*, *j*)

**class** euporie.core.ft.html.**Direction**(*x: bool = False*, *y: bool = False*)

Bases: NamedTuple

A description of a direction.

**count**(*value*, */*)

Return number of occurrences of value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**x: bool**

Alias for field number 0

**y: bool**

Alias for field number 1

**class** euporie.core.ft.html.**HTML**(*markup: str*, *base: Path | str | None = None*, *width: int | None = None*, *height: int | None = None*, *collapse_root_margin: bool = False*, *fill: bool = True*, *css: CssSelectors | None = None*, *browser_css: CssSelectors | None = None*, *mouse_handler: Callable[[Node, MouseEvent], NotImplementedOrNone] | None = None*, *paste_fixed: bool = True*, *defer_assets: bool = False*, *on_update: Callable[[HTML], None] | None = None*, *on_change: Callable[[HTML], None] | None = None*, *_initial_format: str = ''*)

Bases: object

A HTML formatted text renderer.

Accepts a HTML string and renders it at a given width.

**async format_element**(*ft: StyleAndTextTuples*, *element: Node*, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Format an element's content based on its theme.

**async load_assets**() → None

Load remote assets asynchronously.

**property parser: *CustomHTMLParser***

Load the HTML parser.

**process_dom**() → None

Load CSS styles and image resources.

Do not touch element's themes!

**render**(*width: int | None*, *height: int | None*) → StyleAndTextTuples

Render the current markup at a given size.

**async render_details_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render details, showing summary at the top and hiding contents if closed.

**async render_element**(*element:* Node, *available_width: int*, *available_height: int*, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render a Node.

**async render_grid_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render a element with `display` set to `grid`.

**Parameters**

- **element** – The list of parsed elements to render
- **left** – The position on the current line at which to render the output - used to indent subsequent lines when rendering inline blocks like images
- **fill** – Whether to fill the remainder of the rendered space with whitespace
- **align_content** – Whether to align the element's content

**Returns**

Formatted text

**async render_img_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render an image's content.

**async render_input_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render an input element.

**async render_latex_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render LaTeX math content.

**async render_list_item_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render a list item.

**async render_node_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Generate flows for the contents of the element.

**async render_ol_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render lists, adding item numbers to child <li> elements.

**async render_svg_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Display images rendered as ANSI art.

**async render_table_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render a HTML table element.

**Parameters**

- **element** – The list of parsed elements to render
- **left** – The position on the current line at which to render the output - used to indent subsequent lines when rendering inline blocks like images
- **fill** – Whether to fill the remainder of the rendered space with whitespace
- **align_content** – Whether to align the element's content

**Returns**
Formatted text

**async render_text_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render a text element.

**Parameters**

- **element** – The page element to render
- **left** – The position on the current line at which to render the output - used to indent subsequent lines when rendering inline blocks like images
- **fill** – Whether to fill the remainder of the rendered space with whitespace
- **align_content** – Whether to align the element's content

**Returns**
Formatted text

**async render_ul_content**(*element:* Node, *left: int = 0*, *fill: bool = True*, *align_content: bool = True*) → StyleAndTextTuples

Render lists, adding item numbers to child <li> elements.

**property soup: *Node***

Parse the markup.

**class** euporie.core.ft.html.**Node**(*dom:* HTML, *name: str*, *parent:* euporie.core.ft.html.Node | *None*, *text: str = ''*, *attrs: list[tuple[str, str | None]] | None = None*, *contents: list[*euporie.core.ft.html.Node*] | None = None*)

Bases: object

Represent an node in the DOM.

**property child_elements: Generator[*Node*, None, None]**

Yield all of the child element nodes.

**property descendents: Generator[*Node*, None, None]**

Yield all descendent elements.

**find_all**(*tag: str*, *recursive: bool = False*) → Iterator[*Node*]

Find all child elements of a given tag type.

**property first_child_element: *euporie.core.ft.html.Node* | None**

Return the first child element.

**property is_first_child_element: bool**
True if the element if the first child element of its parent element.

**property is_first_child_node: bool**
True if the element if the first child node of its parent element.

**property is_last_child_element: bool**
True if the element if the last child element of its parent element.

**property is_last_child_node: bool**
True if the element if the last child node of its parent element.

**property last_child_element: *euporie.core.ft.html.Node* | None**
Return the last child element.

**property next_element: *euporie.core.ft.html.Node* | None**
Return the next sibling element.

**property next_node: *euporie.core.ft.html.Node* | None**
Return the next sibling node.

**property next_node_in_flow: *euporie.core.ft.html.Node* | None**
Get the next node in the current element's flow.

**property parents: list[*euporie.core.ft.html.Node*]**
Yield all parent elements.

**property preceding_text: str**
Return the text preceding this element.

**property prev_element: *euporie.core.ft.html.Node* | None**
Return the previous sibling element.

**property prev_node: *euporie.core.ft.html.Node* | None**
Return the previous sibling node.

**property prev_node_in_flow: *euporie.core.ft.html.Node* | None**
Get the previous node in the current element's flow.

**property renderable_contents: list[*euporie.core.ft.html.Node*]**
List the node's contents including '::before' and '::after' elements.

**property renderable_descendents: Generator[*Node*, None, None]**
Yield descendents, including pseudo and skipping inline elements.

**reset**() → None
Reset the node and all its children.

**property sibling_element_index: int | None**
Return the index of this element among its siblings.

**property sibling_flow_index: int | None**
Return the index of this element among its siblings.

**property text: str**
Get the element's computed text.

**theme: *Theme***

**class** euporie.core.ft.html.**Theme**(*element:* Node, *parent_theme:* euporie.core.ft.html.Theme | *None*, *available_width: int = 0*, *available_height: int = 0*)

Bases: *Mapping*

The computed theme of an element.

**property anchors: *DiBool***

Which position directions are set.

**property attributes_theme: dict[str, str]**

Calculate the theme defined by (depreciated) HTML attributes.

**property background_color: str**

Get the computed theme background color.

**property base_margin: *DiInt***

Calculate the margin box.

**property block_align: *FormattedTextAlign***

Determine if the left and right margins are set to auto.

**property border_collapse: bool**

Determine if the border is collapsed.

**property border_grid: *GridStyle***

Calculate a GridStyle based on the border lines.

**property border_line: *DiLineStyle***

Calculate the line style.

**property border_style: *DiStr***

Calculate the visibility of the element's borders.

**property border_visibility: *DiBool***

Calculate the visibility of the element's borders.

**property browser_css_theme: dict[str, str]**

Calculate the theme defined in the browser CSS.

**property color: str**

Get the computed theme foreground color.

**property content_height: int**

Return the height available for rendering the element's content.

**property content_width: int**

Return the width available for rendering the element's content.

**property d_block: bool**

If the element a block element.

**property d_blocky: bool**

If the element an inline element.

**property d_flex: bool**

If the element a block element.

**property d_grid: bool**

If the element a block element.

**property d_image: bool**

If the element is an image.

**property d_inline: bool**

If the element an inline element.

**property d_inline_block: bool**

If the element an inline element.

**property d_list_item: bool**

If the element an inline element.

**property d_table: bool**

If the element a block element.

**property d_table_cell: bool**

If the element a block element.

**property dom_css_theme: dict[str, str]**

Calculate the theme defined in CSS in the DOM.

**property floated: str | None**

The float status of the element.

**property font_size: float | int**

Get the computed font size for the current element.

**property gap: tuple[int, int]**

Calculate the horizontal & vertical inter-element spacing.

**get**(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

**property grid_area: str | None**

The name of the grid area assigned to the grid item.

**property grid_areas: dict[int, dict[int, str]]**

The layout of grid-areas in the current node's grid layout.

**property grid_column_span: int**

The number of grid columns spanned by the grid item.

**property grid_column_start: int | None**

The index of the first grid column spanned by the grid item.

**property grid_template: list[list[str]]**

Calculate the size of the grid tracks.

**property height: int | None**

The perscribed height.

**property hidden: bool**

Determine if the element is hidden.

**property in_flow: bool**

Determine if the element is "in-flow".

**property inherited_browser_css_theme: dict[str, str]**

Get the inherited parts from the browser CSS.

**property inherited_theme: dict[str, str]**
Calculate the theme inherited from the element's parent.

**items**() → a set-like object providing a view on D's items

**keys**() → a set-like object providing a view on D's keys

**property latex: bool**
If the element should be rendered as LaTeX.

**property list_style_position: str**
Where the list bullet should be located.

**property list_style_type: str**
The bullet character to use for the list.

**property margin: *DiInt***
Calculate the margin box.

**property max_content_width: int**
Get maximum absolute child width.

**property max_height: int | None**
The maximum permitted height.

**property max_width: int | None**
The maximum permitted width.

**property min_content_width: int**
Get maximum absolute child width.

**property min_height: int | None**
The minimum permitted height.

**property min_width: int | None**
The minimum permitted width.

**property order: tuple[tuple[bool, int], int, tuple[bool, int]]**
Items are sorted by ascending order value then their source code order.

**property padding: *DiInt***
Calculate the padding box.

**property position: *DiInt***
The position of an element with a relative, absolute or fixed position.

**property preformatted: bool**
Determine if the content is pre-formatted.

**reset**() → None
Reset all cached properties.

**property skip: bool**
Determine if the element should not be displayed.

**property style: str**
Calculate the output style.

**property style_attribute_theme: dict[str, str]**

Calculate the theme defined by the element's style attribute.

**property text_align: *FormattedTextAlign***

The text alignment direction.

**async text_transform**(*value: str*) → str

Return a function which transforms text.

**property theme: dict[str, str]**

Return the combined computed theme.

**update_space**(*available_width: int*, *available_height: int*) → None

Set the space available to the element for rendering.

**values**() → an object providing a view on D's values

**property vertical_align: float**

The vertical alignment direction.

**property width: int | None**

The pescribed width.

**property z_index: int**

The z-index of the element.

euporie.core.ft.html.**css_dimension**(*value: str*, *vertical: bool = False*, *available: float | int | None = None*) → float | None

Convert CSS dimensions to terminal cell sizes.

euporie.core.ft.html.**get_color**(*value: str*) → str

Extract a hex color from a string.

euporie.core.ft.html.**get_integer**(*value: str*) → int | None

Extract the first integer from a string.

euporie.core.ft.html.**match_css_selector**(*selector: str*, *attrs: str*, *pseudo: str*, *element_name: str*, *is_first_child_element: bool*, *is_last_child_element: bool*, *sibling_element_index: int | None*, ***element_attrs: Any*) → bool

Determine if a CSS selector matches a particular element.

euporie.core.ft.html.**parse_css_content**(*content: str*) → dict[str, str]

Convert CSS declarations into the internals style representation.

euporie.core.ft.html.**parse_media_condition**(*condition: str*, *dom: HTML*) → *Filter*

Convert media rules to conditions.

euporie.core.ft.html.**parse_style_sheet**(*css_str: str*, *dom: HTML*, *condition: Filter = <prompt_toolkit.filters.base.Always object>*) → None

Collect all CSS styles from style tags.

euporie.core.ft.html.**selector_specificity**(*selector_parts: tuple[*euporie.core.ft.html.CssSelector, *...]*) → tuple[int, int, int]

Calculate the specificity score of a CSS selector.

euporie.core.ft.html.**try_eval**(*value: str*, *default: Any = None*) → Any

Attempt to cast a string to a python type.

## euporie.core.ft.table

Allow drawing tables as `FormattedText`.

### Functions

| | |
|---|---|
| *align*(ft[, how, width, style, placeholder, ...]) | Align formatted text at a given width. |
| *calculate_cell_width*(cell[, render_count]) | Compute the final width of a cell, including padding. |
| *calculate_col_widths*(cols, width, ...[, ...]) | Calculate column widths given the available space. |
| *cast*(typ, val) | Cast a value to a type. |
| *compute_align*(cell[, render_count]) | Compute the alignment of a cell. |
| *compute_border_line*(cell[, render_count]) | Compute a cell's border line. |
| *compute_border_style*(cell[, render_count]) | Compute the cell's final style for each of a cell's borders. |
| *compute_border_visibility*(cell[, render_count]) | Compute a cell's border visibility. |
| *compute_border_width*(cell[, render_count]) | Compute the width of a cell's borders. |
| *compute_lines*(cell, width[, render_count]) | Wrap the cell's text to a given width. |
| *compute_padding*(cell[, render_count]) | Compute a cell's padding. |
| *compute_style*(cell[, render_count]) | Compute a cell's style string. |
| *compute_text*(cell[, render_count]) | Compute a cell's input, converted to `FormattedText`. |
| *fragment_list_width*(fragments) | Return the character width of this text fragment list. |
| *get_app_session*() | |
| *get_grid_char*(key) | Return the character represented by a combination of `LineStyles`. |
| *get_horizontal_edge*(n_bl, s_bl) | Calculate which character to use to divide horizontally adjacent cells. |
| *get_node*(nw_bl, ne_bl, se_bl, sw_bl) | Calculate which character to use at the intersection of four cells. |
| *get_vertical_edge*(w_bl, e_bl) | Calculate which character to use to divide vertically adjacent cells. |
| *join_lines*(fragments) | Join a list of lines of formatted text. |
| *lru_cache*([maxsize, typed]) | Least-recently-used cache decorator. |
| *max_line_width*(ft) | Calculate the length of the longest line in formatted text. |
| *pairwise*(iterable) | Return successiver overlapping pairs from an iterable. |
| *split_lines*(fragments) | Take a single list of (style_str, text) tuples and yield one such list for each line. |
| *tee*(iterable[, n]) | Returns a tuple of n independent iterators. |
| *to_dimension*(value) | Turn the given object into a *Dimension* object. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |
| *to_plain_text*(value) | Turn any kind of formatted text back into plain text. |
| *wrap*(ft, width[, style, placeholder, left, ...]) | Wrap formatted text at a given width. |

### euporie.core.ft.table.align

euporie.core.ft.table.**align**(*ft: StyleAndTextTuples*, *how:* FormattedTextAlign = *FormattedTextAlign.LEFT*, *width: int | None = None*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Align formatted text at a given width.

**Parameters**

- **how** – The alignment direction
- **ft** – The formatted text to strip
- **width** – The width to which the output should be padded. If `None`, the length of the longest line is used
- **style** – The style to apply to the padding
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – If True, whitespace will be ignored

**Returns**

The aligned formatted text

### euporie.core.ft.table.calculate_cell_width

euporie.core.ft.table.**calculate_cell_width**(*cell:* Cell, *render_count: int = 0*) → int

Compute the final width of a cell, including padding.

### euporie.core.ft.table.calculate_col_widths

euporie.core.ft.table.**calculate_col_widths**(*cols: tuple[*euporie.core.ft.table.Col*]*, *width:* Dimension, *expand_to_width: bool*, *min_col_width: int = 2*, *render_count: int = 0*) → list[int]

Calculate column widths given the available space.

Reduce the widest column until we fit in available width, or expand cells to to fill the available width.

**Parameters**

- **cols** – A list of columns in the table
- **width** – The desired width of the table
- **expand_to_width** – Whether the column should expand to fill the available width
- **min_col_width** – The minimum width allowed for a column
- **render_count** – The number of times the app has been rendered

**Returns**

List of new column widths

### euporie.core.ft.table.cast

euporie.core.ft.table.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.ft.table.compute_align

euporie.core.ft.table.**compute_align**(*cell:* Cell, *render_count:* *int* = *0*) → *FormattedTextAlign*

Compute the alignment of a cell.

### euporie.core.ft.table.compute_border_line

euporie.core.ft.table.**compute_border_line**(*cell:* Cell, *render_count:* *int* = *0*) → *DiLineStyle*

Compute a cell's border line.

### euporie.core.ft.table.compute_border_style

euporie.core.ft.table.**compute_border_style**(*cell:* Cell, *render_count:* *int* = *0*) → *DiStr*

Compute the cell's final style for each of a cell's borders.

### euporie.core.ft.table.compute_border_visibility

euporie.core.ft.table.**compute_border_visibility**(*cell:* Cell, *render_count:* *int* = *0*) → *DiBool*

Compute a cell's border visibility.

### euporie.core.ft.table.compute_border_width

euporie.core.ft.table.**compute_border_width**(*cell:* Cell, *render_count:* *int* = *0*) → *DiInt*

Compute the width of a cell's borders.

### euporie.core.ft.table.compute_lines

euporie.core.ft.table.**compute_lines**(*cell:* Cell, *width:* *int*, *render_count:* *int* = *0*) → list[StyleAndTextTuples]

Wrap the cell's text to a given width.

**Parameters**

- **cell** – The cell whose lines to compute
- **width** – The width at which to wrap the cell's text.
- **render_count** – The number of times the application has been rendered

**Returns**

A list of lines of formatted text

### euporie.core.ft.table.compute_padding

euporie.core.ft.table.**compute_padding**(*cell:* Cell, *render_count:* *int* = *0*) → *DiInt*

Compute a cell's padding.

### euporie.core.ft.table.compute_style

euporie.core.ft.table.**compute_style**(*cell:* Cell, *render_count:* *int* = *0*) → str

Compute a cell's style string.

### euporie.core.ft.table.compute_text

euporie.core.ft.table.**compute_text**(*cell:* Cell, *render_count:* *int* = *0*) → StyleAndTextTuples

Compute a cell's input, converted to `FormattedText`.

### euporie.core.ft.table.fragment_list_width

euporie.core.ft.table.**fragment_list_width**(*fragments: StyleAndTextTuples*) → int

Return the character width of this text fragment list.

Takes double width characters into account, and ignore special fragments: * ZeroWidthEscape * ReverseOverwrite

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

**Returns**

The width of the fragment list

### euporie.core.ft.table.get_app_session

euporie.core.ft.table.**get_app_session**() → AppSession

### euporie.core.ft.table.get_grid_char

euporie.core.ft.table.**get_grid_char**(*key:* GridChar) → str

Return the character represented by a combination of `LineStyles`.

### euporie.core.ft.table.get_horizontal_edge

euporie.core.ft.table.**get_horizontal_edge**(*n_bl:* DiLineStyle, *s_bl:* DiLineStyle) → str

Calculate which character to use to divide horizontally adjacent cells.

### euporie.core.ft.table.get_node

euporie.core.ft.table.**get_node**(*nw_bl:* DiLineStyle, *ne_bl:* DiLineStyle, *se_bl:* DiLineStyle, *sw_bl:* DiLineStyle) → str

Calculate which character to use at the intersection of four cells.

### euporie.core.ft.table.get_vertical_edge

euporie.core.ft.table.**get_vertical_edge**(*w_bl:* DiLineStyle, *e_bl:* DiLineStyle) → str

Calculate which character to use to divide vertically adjacent cells.

### euporie.core.ft.table.join_lines

euporie.core.ft.table.**join_lines**(*fragments: list[StyleAndTextTuples]*) → StyleAndTextTuples

Join a list of lines of formatted text.

### euporie.core.ft.table.lru_cache

euporie.core.ft.table.**lru_cache**(*maxsize=128*, *typed=False*)

Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

### euporie.core.ft.table.max_line_width

euporie.core.ft.table.**max_line_width**(*ft: StyleAndTextTuples*) → int

Calculate the length of the longest line in formatted text.

### euporie.core.ft.table.pairwise

euporie.core.ft.table.**pairwise**(*iterable: Iterable[PairT]*) → Iterator[tuple[PairT, PairT]]

Return successiver overlapping pairs from an iterable.

### euporie.core.ft.table.split_lines

euporie.core.ft.table.**split_lines**(*fragments: Iterable[OneStyleAndTextTuple]*) → Iterable[StyleAndTextTuples]

Take a single list of (style_str, text) tuples and yield one such list for each line. Just like str.split, this will yield at least one item.

**Parameters**

**fragments** – Iterable of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.ft.table.tee

euporie.core.ft.table.**tee**(*iterable*, *n=2*, */*)

Returns a tuple of n independent iterators.

### euporie.core.ft.table.to_dimension

euporie.core.ft.table.**to_dimension**(*value: Union[None, int,* Dimension*, Callable[[], Any]]*) → *Dimension*

Turn the given object into a *Dimension* object.

### euporie.core.ft.table.to_formatted_text

euporie.core.ft.table.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

### euporie.core.ft.table.to_plain_text

euporie.core.ft.table.**to_plain_text**(*value: AnyFormattedText*) → str

Turn any kind of formatted text back into plain text.

### euporie.core.ft.table.wrap

euporie.core.ft.table.**wrap**(*ft: StyleAndTextTuples*, *width: int*, *style: str = ''*, *placeholder: str = '…'*, *left: int = 0*, *truncate_long_words: bool = True*, *strip_trailing_ws: bool = False*, *margin: str = ''*) → StyleAndTextTuples

Wrap formatted text at a given width.

If words are longer than the given line they will be truncated

**Parameters**

- **ft** – The formatted text to wrap
- **width** – The width at which to wrap the text
- **style** – The style to apply to the truncation placeholder
- **placeholder** – The string that will appear at the end of a truncated line
- **left** – The starting position within the first line
- **truncate_long_words** – If `True` words longer than a line will be truncated
- **strip_trailing_ws** – If `True`, trailing whitespace will be removed from the ends of lines
- **margin** – Text to use a margin for the continuation of wrapped lines

**Returns**

The wrapped formatted text

### Classes

| | |
|---|---|
| *Cell*([text, row, col, colspan, rowspan, ...]) | A table cell. |
| *Col*([table, cells, align, style, padding, ...]) | A column in a table. |
| *DiBool*([top, right, bottom, left]) | A tuple of four bools with directions. |
| *DiInt*([top, right, bottom, left]) | A tuple of four integers with directions. |
| *DiLineStyle*([top, right, bottom, left]) | A description of a cell border: a *LineStyle* for each edge. |
| *DiStr*([top, right, bottom, left]) | A tuple of four strings with directions. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *DummyCol*([table, cells, align, style, ...]) | A dummy column - created to hold cells without an assigned column. |
| *DummyRow*([table, cells, align, style, ...]) | A dummy row - created to hold cells without an assigned column. |
| *DummyTable*(*args, **kwargs) | A dummy table - created to hold rows and columns without an assigned table. |
| *FormattedTextAlign*(value[, names, module, ...]) | Alignment of formatted text. |
| *GridChar*(north, east, south, west) | Repreentation of a grid node character. |
| *LineStyle*(name, rank[, parent, visible]) | Define a line style which can be used to draw grids. |
| *Row*([table, cells, align, style, padding, ...]) | A row in a table. |
| *RowCol*([table, cells, align, style, ...]) | Base class for table rows and columns. |
| *SpacerCell*(expands, span_row_index, ...[, ...]) | A dummy cell to virtually occupy space when `colspan` or `rowspan` are used. |
| *Table*([rows, cols, width, expand, align, ...]) | A table. |
| *defaultdict* | defaultdict(default_factory=None, /, [...]) --> dict with default factory |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |
| *zip_longest* | zip_longest(iter1 [,iter2 [...]], [fillvalue=None]) --> zip_longest object |

### euporie.core.ft.table.Cell

**class** euporie.core.ft.table.**Cell**(*text: AnyFormattedText = ''*, *row:* Row | *None = None*, *col:* Col | *None = None*, *colspan: int = 1*, *rowspan: int = 1*, *width: int* | *None = None*, *align:* FormattedTextAlign | *None = None*, *style: str = ''*, *padding:* DiInt | *int = 0*, *border_line:* DiLineStyle | LineStyle = *LineStyle(Thin)*, *border_style:* DiStr | *str = ''*, *border_visibility:* DiBool | *bool* | *None = True*)

A table cell.

### euporie.core.ft.table.Col

**class** euporie.core.ft.table.**Col**(*table:* Table | *None* = *None*, *cells: Sequence[*Cell*]* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style: str* = *''*, *padding:* DiInt | *int* = *0*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = *''*, *border_visibility:* DiBool | *bool* | *None* = *None*)

A column in a table.

### euporie.core.ft.table.DiBool

**class** euporie.core.ft.table.**DiBool**(*top: bool* = *False*, *right: bool* = *False*, *bottom: bool* = *False*, *left: bool* = *False*)

A tuple of four bools with directions.

### euporie.core.ft.table.DiInt

**class** euporie.core.ft.table.**DiInt**(*top: int* = *0*, *right: int* = *0*, *bottom: int* = *0*, *left: int* = *0*)

A tuple of four integers with directions.

### euporie.core.ft.table.DiLineStyle

**class** euporie.core.ft.table.**DiLineStyle**(*top:* LineStyle = *LineStyle(None)*, *right:* LineStyle = *LineStyle(None)*, *bottom:* LineStyle = *LineStyle(None)*, *left:* LineStyle = *LineStyle(None)*)

A description of a cell border: a `LineStyle` for each edge.

### euporie.core.ft.table.DiStr

**class** euporie.core.ft.table.**DiStr**(*top: str* = *''*, *right: str* = *''*, *bottom: str* = *''*, *left: str* = *''*)

A tuple of four strings with directions.

### euporie.core.ft.table.Dimension

**class** euporie.core.ft.table.**Dimension**(*min: int* | *None* = *None*, *max: int* | *None* = *None*, *weight: int* | *None* = *None*, *preferred: int* | *None* = *None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.

- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.ft.table.DummyCol

**class** euporie.core.ft.table.**DummyCol**(*table:* Table | *None* = *None*, *cells: Sequence[*Cell*]* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style: str* = *''*, *padding:* DiInt | *int* = *0*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = *''*, *border_visibility:* DiBool | *bool* | *None* = *None*)

A dummy column - created to hold cells without an assigned column.

### euporie.core.ft.table.DummyRow

**class** euporie.core.ft.table.**DummyRow**(*table:* Table | *None* = *None*, *cells: Sequence[*Cell*]* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style: str* = *''*, *padding:* DiInt | *int* = *0*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = *''*, *border_visibility:* DiBool | *bool* | *None* = *None*)

A dummy row - created to hold cells without an assigned column.

### euporie.core.ft.table.DummyTable

**class** euporie.core.ft.table.**DummyTable**(**args: Any*, ***kwargs: Any*)

A dummy table - created to hold rows and columns without an assigned table.

### euporie.core.ft.table.FormattedTextAlign

**class** euporie.core.ft.table.**FormattedTextAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of formatted text.

### euporie.core.ft.table.GridChar

**class** euporie.core.ft.table.**GridChar**(*north:* LineStyle, *east:* LineStyle, *south:* LineStyle, *west:* LineStyle)

Repreentation of a grid node character.

The four compass points represent the line style joining from the given direction.

### euporie.core.ft.table.LineStyle

**class** euporie.core.ft.table.**LineStyle**(*name: str*, *rank: tuple[int, int]*, *parent:* euporie.core.border.LineStyle | *None* = *None*, *visible: bool* = *True*)

Define a line style which can be used to draw grids.

`GridStyle`s can be created from a :class:`LineStyle` by accessing an attribute with the name of a default mask from `Masks`.

### euporie.core.ft.table.Row

**class** euporie.core.ft.table.**Row**(*table:* Table | *None* = *None*, *cells: Sequence[*Cell*]* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style: str* = '', *padding:* DiInt | *int* = *0*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = '', *border_visibility:* DiBool | *bool* | *None* = *None*)

A row in a table.

### euporie.core.ft.table.RowCol

**class** euporie.core.ft.table.**RowCol**(*table:* Table | *None* = *None*, *cells: Sequence[*Cell*]* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style: str* = '', *padding:* DiInt | *int* = *0*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = '', *border_visibility:* DiBool | *bool* | *None* = *None*)

Base class for table rows and columns.

### euporie.core.ft.table.SpacerCell

**class** euporie.core.ft.table.**SpacerCell**(*expands:* Cell, *span_row_index: int*, *span_col_index: int*, *text: AnyFormattedText* = '', *row:* Row | *None* = *None*, *col:* Col | *None* = *None*, *colspan: int* = *1*, *rowspan: int* = *1*, *width: int* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style: str* = '', *padding:* DiInt | *int* | *None* = *None*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = '', *border_visibility:* DiBool | *bool* | *None* = *None*)

A dummy cell to virtually occupy space when `colspan` or `rowspan` are used.

### euporie.core.ft.table.Table

**class** euporie.core.ft.table.**Table**(*rows: Sequence[Row] | None = None*, *cols: Sequence[Col] | None = None*, *width: AnyDimension | None = None*, *expand: bool = False*, *align: FormattedTextAlign = FormattedTextAlign.LEFT*, *style: str = ''*, *padding: DiInt | int | None = None*, *border_line: DiLineStyle | LineStyle = LineStyle(None)*, *border_style: DiStr | str = ''*, *border_visibility: DiBool | bool = False*, *background_style: str = ''*)

A table.

### euporie.core.ft.table.defaultdict

**class** euporie.core.ft.table.**defaultdict**

defaultdict(default_factory=None, /, […]) –> dict with default factory

The default factory is called without arguments to produce a new value when a key is not present, in __getitem__ only. A defaultdict compares equal to a dict with the same items. All remaining arguments are treated the same as if they were passed to the dict constructor, including keyword arguments.

### euporie.core.ft.table.partial

**class** euporie.core.ft.table.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

### euporie.core.ft.table.zip_longest

**class** euporie.core.ft.table.**zip_longest**

zip_longest(iter1 [,iter2 […]], [fillvalue=None]) –> zip_longest object

Return a zip_longest object whose .__next__() method returns a tuple where the i-th element comes from the i-th iterable argument. The .__next__() method continues until the longest iterable in the argument sequence is exhausted and then it raises StopIteration. When the shorter iterables are exhausted, the fillvalue is substituted in their place. The fillvalue defaults to None or can be specified by a keyword argument.

**class** euporie.core.ft.table.**Cell**(*text: AnyFormattedText = ''*, *row: Row | None = None*, *col: Col | None = None*, *colspan: int = 1*, *rowspan: int = 1*, *width: int | None = None*, *align: FormattedTextAlign | None = None*, *style: str = ''*, *padding: DiInt | int = 0*, *border_line: DiLineStyle | LineStyle = LineStyle(Thin)*, *border_style: DiStr | str = ''*, *border_visibility: DiBool | bool | None = True*)

Bases: object

A table cell.

**property border_line: *DiLineStyle***

The cell's border line.

**property border_style: *DiStr***

The cell's border style.

**property padding: *DiInt***

The cell's padding.

**class** euporie.core.ft.table.**Col**(*table:* Table | *None* = *None*, *cells:* *Sequence[*Cell*]* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style:* *str* = *''*, *padding:* DiInt | *int* = *0*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = *''*, *border_visibility:* DiBool | *bool* | *None* = *None*)

Bases: *RowCol*

A column in a table.

**add_cell**(*cell:* Cell, *index:* *int* | *None* = *None*) → None

Add a cell to the row/ column.

**property border_line: *DiLineStyle***

The cell's border line.

**property border_style: *DiStr***

The cell's border style.

**property cells: list[*euporie.core.ft.table.Cell*]**

List the cells in the row/column.

**new_cell**(**args: Any*, ***kwargs: Any*) → *Cell*

Create a new cell in this row/column.

**property padding: *DiInt***

The cell's padding.

**span_type: str = 'rowspan'**

**class** euporie.core.ft.table.**DummyCol**(*table:* Table | *None* = *None*, *cells:* *Sequence[*Cell*]* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style:* *str* = *''*, *padding:* DiInt | *int* = *0*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = *''*, *border_visibility:* DiBool | *bool* | *None* = *None*)

Bases: *Col*

A dummy column - created to hold cells without an assigned column.

**add_cell**(*cell:* Cell, *index:* *int* | *None* = *None*) → None

Prevent cells being added to a dummy column.

**property border_line: *DiLineStyle***

The cell's border line.

**property border_style: *DiStr***

The cell's border style.

**property cells: list[*euporie.core.ft.table.Cell*]**

List the cells in the row/column.

**new_cell**(**args: Any*, ***kwargs: Any*) → *Cell*

Create a new cell in this row/column.

**property padding: *DiInt***

The cell's padding.

**span_type: str = 'rowspan'**

**class** euporie.core.ft.table.**DummyRow**(*table:* Table | *None* = *None*, *cells: Sequence[*Cell*]* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style: str* = *''*, *padding:* DiInt | *int* = *0*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = *''*, *border_visibility:* DiBool | *bool* | *None* = *None*)

Bases: *Row*

A dummy row - created to hold cells without an assigned column.

**add_cell**(*cell:* Cell, *index: int* | *None* = *None*) → None

Prevent cells being added to a dummy row.

**property border_line: *DiLineStyle***

The cell's border line.

**property border_style: *DiStr***

The cell's border style.

**property cells: list[*euporie.core.ft.table.Cell*]**

List the cells in the row/column.

**new_cell**(**args: Any*, ***kwargs: Any*) → *Cell*

Create a new cell in this row/column.

**property padding: *DiInt***

The cell's padding.

**span_type: str = 'colspan'**

**class** euporie.core.ft.table.**DummyTable**(**args: Any*, ***kwargs: Any*)

Bases: *Table*

A dummy table - created to hold rows and columns without an assigned table.

**add_col**(*col:* Col) → None

Prevent columns being added to a dummy table.

**add_row**(*row:* Row) → None

Prevent rows being added to a dummy table.

**property border_line: *DiLineStyle***

The cell's border line.

**property border_style: *DiStr***

The cell's border style.

**calculate_cell_widths**(*width: AnyDimension* | *None* = *None*) → dict[*Cell*, int]

Calculate widths for each table cell, taking colspans into account.

**calculate_col_widths**(*width: AnyDimension* | *None* = *None*, *min_col_width: int* = *4*) → list[int]

Calculate the table's column widths.

**property cols: list[*euporie.core.ft.table.Col*]**

A list of columns in the table.

**draw_table_row**(*row_above:* RowCol | *None*, *row_below:* RowCol | *None*, *cell_widths: dict[*Cell*, int]*, *col_widths: list[int]*, *row_edge_visibility: bool*, *col_edge_visibilities: dict[int, bool]*) → Iterable[StyleAndTextTuples]

Draw a row in the table.

**new_col**(**args: Any*, ***kwargs: Any*) → *Col*

Create a new column in the table.

**new_row**(**args: Any*, ***kwargs: Any*) → *Row*

Create a new row in the table.

**property padding: *DiInt***

The cell's padding.

**render**(*width: AnyDimension | None = None*) → StyleAndTextTuples

Draw the table, optionally at a given character width.

**property rows: list[*euporie.core.ft.table.Row*]**

A list of rows in the table.

**sync_cols_to_rows**() → None

Enure cells in columns are present in the relevant rows.

**sync_rows_to_cols**() → None

Enure cells in rows are present in the relevant columns.

**property width: *Dimension***

The table's width.

**class** euporie.core.ft.table.**Row**(*table: Table | None = None*, *cells: Sequence[Cell] | None = None*, *align: FormattedTextAlign | None = None*, *style: str = ''*, *padding: DiInt | int = 0*, *border_line: DiLineStyle | LineStyle = LineStyle(None)*, *border_style: DiStr | str = ''*, *border_visibility: DiBool | bool | None = None*)

Bases: *RowCol*

A row in a table.

**add_cell**(*cell: Cell*, *index: int | None = None*) → None

Add a cell to the row/ column.

**property border_line: *DiLineStyle***

The cell's border line.

**property border_style: *DiStr***

The cell's border style.

**property cells: list[*euporie.core.ft.table.Cell*]**

List the cells in the row/column.

**new_cell**(**args: Any*, ***kwargs: Any*) → *Cell*

Create a new cell in this row/column.

**property padding: *DiInt***

The cell's padding.

**span_type: str = 'colspan'**

**class** euporie.core.ft.table.**RowCol**(*table: Table | None = None*, *cells: Sequence[Cell] | None = None*, *align: FormattedTextAlign | None = None*, *style: str = ''*, *padding: DiInt | int = 0*, *border_line: DiLineStyle | LineStyle = LineStyle(None)*, *border_style: DiStr | str = ''*, *border_visibility: DiBool | bool | None = None*)

Bases: object

Base class for table rows and columns.

**add_cell**(*cell:* Cell, *index:* *int* | *None* = *None*) → None

Add a cell to the row/ column.

**property border_line:** ***DiLineStyle***

The cell's border line.

**property border_style:** ***DiStr***

The cell's border style.

**property cells:** **list[*euporie.core.ft.table.Cell*]**

List the cells in the row/column.

**new_cell**(**args: Any*, ***kwargs: Any*) → *Cell*

Create a new cell in this row/column.

**property padding:** ***DiInt***

The cell's padding.

**span_type:** **str**

**class** euporie.core.ft.table.**SpacerCell**(*expands:* Cell, *span_row_index:* *int*, *span_col_index:* *int*, *text: AnyFormattedText = ''*, *row:* Row | *None* = *None*, *col:* Col | *None* = *None*, *colspan:* *int* = *1*, *rowspan:* *int* = *1*, *width:* *int* | *None* = *None*, *align:* FormattedTextAlign | *None* = *None*, *style:* *str* = *''*, *padding:* DiInt | *int* | *None* = *None*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = *''*, *border_visibility:* DiBool | *bool* | *None* = *None*)

Bases: *Cell*

A dummy cell to virtually occupy space when `colspan` or `rowspan` are used.

**property border_line:** ***DiLineStyle***

The cell's border line.

**property border_style:** ***DiStr***

The cell's border style.

**property padding:** ***DiInt***

The cell's padding.

**class** euporie.core.ft.table.**Table**(*rows: Sequence[Row]* | *None* = *None*, *cols: Sequence[Col]* | *None* = *None*, *width: AnyDimension* | *None* = *None*, *expand:* *bool* = *False*, *align:* FormattedTextAlign = *FormattedTextAlign.LEFT*, *style:* *str* = *''*, *padding:* DiInt | *int* | *None* = *None*, *border_line:* DiLineStyle | LineStyle = *LineStyle(None)*, *border_style:* DiStr | *str* = *''*, *border_visibility:* DiBool | *bool* = *False*, *background_style:* *str* = *''*)

Bases: object

A table.

**add_col**(*col:* Col) → None

Add a column to the table.

**add_row**(*row:* Row) → None

Add a row to the table.

**property border_line: *DiLineStyle***

The cell's border line.

**property border_style: *DiStr***

The cell's border style.

**calculate_cell_widths**(*width: AnyDimension | None = None*) → dict[*Cell*, int]

Calculate widths for each table cell, taking colspans into account.

**calculate_col_widths**(*width: AnyDimension | None = None*, *min_col_width: int = 4*) → list[int]

Calculate the table's column widths.

**property cols: list[*euporie.core.ft.table.Col*]**

A list of columns in the table.

**draw_table_row**(*row_above:* RowCol | *None*, *row_below:* RowCol | *None*, *cell_widths: dict[*Cell, *int]*, *col_widths: list[int]*, *row_edge_visibility: bool*, *col_edge_visibilities: dict[int, bool]*) → Iterable[StyleAndTextTuples]

Draw a row in the table.

**new_col**(**args: Any*, ***kwargs: Any*) → *Col*

Create a new column in the table.

**new_row**(**args: Any*, ***kwargs: Any*) → *Row*

Create a new row in the table.

**property padding: *DiInt***

The cell's padding.

**render**(*width: AnyDimension | None = None*) → StyleAndTextTuples

Draw the table, optionally at a given character width.

**property rows: list[*euporie.core.ft.table.Row*]**

A list of rows in the table.

**sync_cols_to_rows**() → None

Enure cells in columns are present in the relevant rows.

**sync_rows_to_cols**() → None

Enure cells in rows are present in the relevant columns.

**property width: *Dimension***

The table's width.

euporie.core.ft.table.**calculate_cell_width**(*cell:* Cell, *render_count: int = 0*) → int

Compute the final width of a cell, including padding.

euporie.core.ft.table.**calculate_col_widths**(*cols: tuple[*euporie.core.ft.table.Col*]*, *width:* Dimension, *expand_to_width: bool*, *min_col_width: int = 2*, *render_count: int = 0*) → list[int]

Calculate column widths given the available space.

Reduce the widest column until we fit in available width, or expand cells to to fill the available width.

**Parameters**

- **cols** – A list of columns in the table
- **width** – The desired width of the table
- **expand_to_width** – Whether the column should expand to fill the available width
- **min_col_width** – The minimum width allowed for a column
- **render_count** – The number of times the app has been rendered

**Returns**
List of new column widths

euporie.core.ft.table.**compute_align**(*cell:* Cell, *render_count:* *int* = *0*) → *FormattedTextAlign*
Compute the alignment of a cell.

euporie.core.ft.table.**compute_border_line**(*cell:* Cell, *render_count:* *int* = *0*) → *DiLineStyle*
Compute a cell's border line.

euporie.core.ft.table.**compute_border_style**(*cell:* Cell, *render_count:* *int* = *0*) → *DiStr*
Compute the cell's final style for each of a cell's borders.

euporie.core.ft.table.**compute_border_visibility**(*cell:* Cell, *render_count:* *int* = *0*) → *DiBool*
Compute a cell's border visibility.

euporie.core.ft.table.**compute_border_width**(*cell:* Cell, *render_count:* *int* = *0*) → *DiInt*
Compute the width of a cell's borders.

euporie.core.ft.table.**compute_lines**(*cell:* Cell, *width:* *int*, *render_count:* *int* = *0*) → list[StyleAndTextTuples]
Wrap the cell's text to a given width.

**Parameters**

- **cell** – The cell whose lines to compute
- **width** – The width at which to wrap the cell's text.
- **render_count** – The number of times the application has been rendered

**Returns**
A list of lines of formatted text

euporie.core.ft.table.**compute_padding**(*cell:* Cell, *render_count:* *int* = *0*) → *DiInt*
Compute a cell's padding.

euporie.core.ft.table.**compute_style**(*cell:* Cell, *render_count:* *int* = *0*) → str
Compute a cell's style string.

euporie.core.ft.table.**compute_text**(*cell:* Cell, *render_count:* *int* = *0*) → StyleAndTextTuples
Compute a cell's input, converted to `FormattedText`.

euporie.core.ft.table.**get_horizontal_edge**(*n_bl:* DiLineStyle, *s_bl:* DiLineStyle) → str
Calculate which character to use to divide horizontally adjacent cells.

euporie.core.ft.table.**get_node**(*nw_bl:* DiLineStyle, *ne_bl:* DiLineStyle, *se_bl:* DiLineStyle, *sw_bl:* DiLineStyle) → str
Calculate which character to use at the intersection of four cells.

euporie.core.ft.table.**get_vertical_edge**(*w_bl:* DiLineStyle, *e_bl:* DiLineStyle) → str
Calculate which character to use to divide vertically adjacent cells.

euporie.core.ft.table.**pairwise**(*iterable: Iterable[PairT]*) → Iterator[tuple[PairT, PairT]]

Return successiver overlapping pairs from an iterable.

### euporie.core.ft.utils

Utilities for manipulating formatted text.

### Functions

| | |
|---|---|
| *add_border*(ft[, width, style, border_grid, ...]) | Add a border around formatted text. |
| *align*(ft[, how, width, style, placeholder, ...]) | Align formatted text at a given width. |
| *apply_reverse_overwrites*(ft) | Write fragments tagged with "[ReverseOverwrite]" over text to their left. |
| *apply_style*(ft, style) | Apply a style to formatted text. |
| *cast*(typ, val) | Cast a value to a type. |
| *concat*(ft_a, ft_b[, baseline_a, baseline_b, ...]) | Concatenate two blocks of formatted text, aligning at a given baseline. |
| *explode_text_fragments*(fragments) | Turn a list of (style_str, text) tuples into another list where each string is exactly one character. |
| *fragment_list_to_text*(fragments) | Concatenate all the text parts again. |
| *fragment_list_to_words*(fragments[, sep]) | Split formatted text into a list of word fragments which form words. |
| *fragment_list_width*(fragments) | Return the character width of this text fragment list. |
| *get_cwidth*(string) | Return width of a string. |
| *get_lexer_by_name*(_alias, **options) | Return an instance of a *Lexer* subclass that has *alias* in its aliases list. |
| *indent*(ft[, margin, style, skip_first]) | Indent formatted text with a given margin. |
| *join_lines*(fragments) | Join a list of lines of formatted text. |
| *last_char*(ft) | Retrieve the last character of formatted text. |
| *lex*(ft, lexer_name) | Format formatted text using a named `pygments` lexer. |
| *max_line_width*(ft) | Calculate the length of the longest line in formatted text. |
| *pad*(ft[, width, char, style]) | Fill space at the end of lines. |
| *paste*(ft_top, ft_bottom[, row, col, transparent]) | Pate formatted text on top of other formatted text. |
| *split_lines*(fragments) | Take a single list of (style_str, text) tuples and yield one such list for each line. |
| *strip*(ft[, left, right, chars, only_unstyled]) | Strip whitespace (or a given character) from the ends of formatted text. |
| *strip_one_trailing_newline*(ft) | Remove up to one trailing new-line character from formatted text. |
| *substring*(ft[, start, end]) | Extract a substring from formatted text. |
| *to_plain_text*(value) | Turn any kind of formatted text back into plain text. |
| *truncate*(ft, width[, style, placeholder, ...]) | Truncate all lines at a given length. |
| *valign*(ft[, how, height, style]) | Align formatted text vertically. |
| *wrap*(ft, width[, style, placeholder, left, ...]) | Wrap formatted text at a given width. |

### euporie.core.ft.utils.add_border

euporie.core.ft.utils.**add_border**(*ft: StyleAndTextTuples*, *width: int | None = None*, *style: str = ''*, *border_grid: GridStyle = [illegible]*, *border_visibility: DiBool | bool = True*, *border_style: DiStr | str = ''*, *padding: DiInt | int = 0*, *padding_style: DiStr | str = ''*) → StyleAndTextTuples

Add a border around formatted text.

**Parameters**

- **ft** – The formatted text to enclose with a border
- **width** – The target width including the border and padding
- **style** – The style to apply to the content background
- **border_grid** – The grid style to use for the border
- **border_visibility** – Determines which edges should receive a border
- **border_style** – The style to apply to the border
- **padding** – The width of spacing to apply between the content and the border
- **padding_style** – The style to apply to the border

**Returns**

The indented formatted text

### euporie.core.ft.utils.align

euporie.core.ft.utils.**align**(*ft: StyleAndTextTuples*, *how: FormattedTextAlign = FormattedTextAlign.LEFT*, *width: int | None = None*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Align formatted text at a given width.

**Parameters**

- **how** – The alignment direction
- **ft** – The formatted text to strip
- **width** – The width to which the output should be padded. If `None`, the length of the longest line is used
- **style** – The style to apply to the padding
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – If True, whitespace will be ignored

**Returns**

The aligned formatted text

### euporie.core.ft.utils.apply_reverse_overwrites

euporie.core.ft.utils.**apply_reverse_overwrites**(*ft: StyleAndTextTuples*) → StyleAndTextTuples

Write fragments tagged with "[ReverseOverwrite]" over text to their left.

### euporie.core.ft.utils.apply_style

euporie.core.ft.utils.**apply_style**(*ft: StyleAndTextTuples*, *style: str*) → StyleAndTextTuples

Apply a style to formatted text.

### euporie.core.ft.utils.cast

euporie.core.ft.utils.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.ft.utils.concat

euporie.core.ft.utils.**concat**(*ft_a: StyleAndTextTuples*, *ft_b: StyleAndTextTuples*, *baseline_a: int = 0*, *baseline_b: int = 0*, *style: str = ''*) → tuple[StyleAndTextTuples, int]

Concatenate two blocks of formatted text, aligning at a given baseline.

**Parameters**

- **ft_a** – The first block of formatted text to combine
- **ft_b** – The second block of formatted text to combine
- **baseline_a** – The row to use to align the first block of formatted text with the second, counted in lines down from the top of the block
- **baseline_b** – The row to use to align the second block of formatted text with the second, counted in lines down from the top of the block
- **style** – The style to use for any extra lines added

**Returns**

A tuple containing the combined formatted text and the new baseline position

### euporie.core.ft.utils.explode_text_fragments

euporie.core.ft.utils.**explode_text_fragments**(*fragments: Iterable[_T]*) → _ExplodedList[_T]

Turn a list of (style_str, text) tuples into another list where each string is exactly one character.

It should be fine to call this function several times. Calling this on a list that is already exploded, is a null operation.

**Parameters**

**fragments** – List of (style, text) tuples.

### euporie.core.ft.utils.fragment_list_to_text

euporie.core.ft.utils.**fragment_list_to_text**(*fragments: StyleAndTextTuples*) → str

Concatenate all the text parts again.

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.ft.utils.fragment_list_to_words

euporie.core.ft.utils.**fragment_list_to_words**(*fragments: StyleAndTextTuples*, *sep: str = ' '*) → Iterable[StyleAndTextTuples]

Split formatted text into a list of word fragments which form words.

### euporie.core.ft.utils.fragment_list_width

euporie.core.ft.utils.**fragment_list_width**(*fragments: StyleAndTextTuples*) → int

Return the character width of this text fragment list.

Takes double width characters into account, and ignore special fragments: * ZeroWidthEscape * ReverseOverwrite

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

**Returns**

The width of the fragment list

### euporie.core.ft.utils.get_cwidth

euporie.core.ft.utils.**get_cwidth**(*string: str*) → int

Return width of a string. Wrapper around `wcwidth`.

### euporie.core.ft.utils.get_lexer_by_name

euporie.core.ft.utils.**get_lexer_by_name**(*_alias*, ***options*)

Return an instance of a *Lexer* subclass that has *alias* in its aliases list. The lexer is given the *options* at its instantiation.

Will raise *pygments.util.ClassNotFound* if no lexer with that alias is found.

### euporie.core.ft.utils.indent

euporie.core.ft.utils.**indent**(*ft: StyleAndTextTuples*, *margin: str = ' '*, *style: str = ''*, *skip_first: bool = False*) → StyleAndTextTuples

Indent formatted text with a given margin.

**Parameters**

- **ft** – The formatted text to strip
- **margin** – The margin string to add
- **style** – The style to apply to the margin
- **skip_first** – If `True`, the first line is skipped

**Returns**

The indented formatted text

### euporie.core.ft.utils.join_lines

euporie.core.ft.utils.**join_lines**(*fragments: list[StyleAndTextTuples]*) → StyleAndTextTuples

Join a list of lines of formatted text.

### euporie.core.ft.utils.last_char

euporie.core.ft.utils.**last_char**(*ft: StyleAndTextTuples*) → str | None

Retrieve the last character of formatted text.

### euporie.core.ft.utils.lex

euporie.core.ft.utils.**lex**(*ft: StyleAndTextTuples*, *lexer_name: str*) → StyleAndTextTuples

Format formatted text using a named `pygments` lexer.

### euporie.core.ft.utils.max_line_width

euporie.core.ft.utils.**max_line_width**(*ft: StyleAndTextTuples*) → int

Calculate the length of the longest line in formatted text.

### euporie.core.ft.utils.pad

euporie.core.ft.utils.**pad**(*ft: StyleAndTextTuples*, *width: int | None = None*, *char: str = ' '*, *style: str = ''*) → StyleAndTextTuples

Fill space at the end of lines.

### euporie.core.ft.utils.paste

euporie.core.ft.utils.**paste**(*ft_top: StyleAndTextTuples*, *ft_bottom: StyleAndTextTuples*, *row: int = 0*, *col: int = 0*, *transparent: bool = False*) → StyleAndTextTuples

Pate formatted text on top of other formatted text.

### euporie.core.ft.utils.split_lines

euporie.core.ft.utils.**split_lines**(*fragments: Iterable[OneStyleAndTextTuple]*) → Iterable[StyleAndTextTuples]

Take a single list of (style_str, text) tuples and yield one such list for each line. Just like str.split, this will yield at least one item.

**Parameters**

**fragments** – Iterable of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.ft.utils.strip

euporie.core.ft.utils.**strip**(*ft: StyleAndTextTuples*, *left: bool = True*, *right: bool = True*, *chars: str | None = None*, *only_unstyled: bool = False*) → StyleAndTextTuples

Strip whitespace (or a given character) from the ends of formatted text.

**Parameters**

- **ft** – The formatted text to strip
- **left** – If `True`, strip from the left side of the input
- **right** – If `True`, strip from the right side of the input
- **chars** – The character to strip. If `None`, strips whitespace
- **only_unstyled** – If `True`, only strip unstyled fragments

**Returns**

The stripped formatted text

### euporie.core.ft.utils.strip_one_trailing_newline

euporie.core.ft.utils.**strip_one_trailing_newline**(*ft: StyleAndTextTuples*) → StyleAndTextTuples

Remove up to one trailing new-line character from formatted text.

### euporie.core.ft.utils.substring

euporie.core.ft.utils.**substring**(*ft: StyleAndTextTuples*, *start: int | None = None*, *end: int | None = None*) → StyleAndTextTuples

Extract a substring from formatted text.

### euporie.core.ft.utils.to_plain_text

euporie.core.ft.utils.**to_plain_text**(*value: AnyFormattedText*) → str

Turn any kind of formatted text back into plain text.

### euporie.core.ft.utils.truncate

euporie.core.ft.utils.**truncate**(*ft: StyleAndTextTuples*, *width: int*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Truncate all lines at a given length.

**Parameters**

- **ft** – The formatted text to truncate
- **width** – The width at which to truncate the text
- **style** – The style to apply to the truncation placeholder. The style of the truncated text will be used if not provided
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – Do not use placeholder when truncating whitespace

**Returns**

The truncated formatted text

### euporie.core.ft.utils.valign

euporie.core.ft.utils.**valign**(*ft: StyleAndTextTuples*, *how: FormattedTextVerticalAlign = FormattedTextVerticalAlign.MIDDLE*, *height: int | None = None*, *style: str = ''*) → StyleAndTextTuples

Align formatted text vertically.

### euporie.core.ft.utils.wrap

euporie.core.ft.utils.**wrap**(*ft: StyleAndTextTuples*, *width: int*, *style: str = ''*, *placeholder: str = '…'*, *left: int = 0*, *truncate_long_words: bool = True*, *strip_trailing_ws: bool = False*, *margin: str = ''*) → StyleAndTextTuples

Wrap formatted text at a given width.

If words are longer than the given line they will be truncated

**Parameters**

- **ft** – The formatted text to wrap

- **width** – The width at which to wrap the text
- **style** – The style to apply to the truncation placeholder
- **placeholder** – The string that will appear at the end of a truncated line
- **left** – The starting position within the first line
- **truncate_long_words** – If `True` words longer than a line will be truncated
- **strip_trailing_ws** – If `True`, trailing whitespace will be removed from the ends of lines
- **margin** – Text to use a margin for the continuation of wrapped lines

**Returns**
The wrapped formatted text

## Classes

| | |
|---|---|
| *DiBool*([top, right, bottom, left]) | A tuple of four bools with directions. |
| *DiInt*([top, right, bottom, left]) | A tuple of four integers with directions. |
| *DiStr*([top, right, bottom, left]) | A tuple of four strings with directions. |
| *Enum*(value[, names, module, qualname, type, ...]) | Create a collection of name/value pairs. |
| *FormattedTextAlign*(value[, names, module, ...]) | Alignment of formatted text. |
| *FormattedTextVerticalAlign*(value[, names, ...]) | Vertical alignment of formatted text. |
| *GridStyle*(line_style, mask) | A collection of characters which can be used to draw a grid. |

### euporie.core.ft.utils.DiBool

**class** euporie.core.ft.utils.**DiBool**(*top: bool = False*, *right: bool = False*, *bottom: bool = False*, *left: bool = False*)

A tuple of four bools with directions.

### euporie.core.ft.utils.DiInt

**class** euporie.core.ft.utils.**DiInt**(*top: int = 0*, *right: int = 0*, *bottom: int = 0*, *left: int = 0*)

A tuple of four integers with directions.

### euporie.core.ft.utils.DiStr

**class** euporie.core.ft.utils.**DiStr**(*top: str = ''*, *right: str = ''*, *bottom: str = ''*, *left: str = ''*)

A tuple of four strings with directions.

### euporie.core.ft.utils.Enum

**class** euporie.core.ft.utils.**Enum**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes – see the documentation for details.

### euporie.core.ft.utils.FormattedTextAlign

**class** euporie.core.ft.utils.**FormattedTextAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of formatted text.

### euporie.core.ft.utils.FormattedTextVerticalAlign

**class** euporie.core.ft.utils.**FormattedTextVerticalAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Vertical alignment of formatted text.

### euporie.core.ft.utils.GridStyle

**class** euporie.core.ft.utils.**GridStyle**(*line_style: ~euporie.core.border.LineStyle = LineStyle(None)*, *mask: ~euporie.core.border.Mask = <euporie.core.border.Mask object>*)

A collection of characters which can be used to draw a grid.

### Exceptions

| | |
|---|---|
| *ClassNotFound* | Raised if one of the lookup functions didn't find a matching class. |

### euporie.core.ft.utils.ClassNotFound

**exception** euporie.core.ft.utils.**ClassNotFound**

Raised if one of the lookup functions didn't find a matching class.

**class** euporie.core.ft.utils.**FormattedTextAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *Enum*

Alignment of formatted text.

**CENTER = 'center'**

**LEFT = 'left'**

**RIGHT = 'right'**

**class** euporie.core.ft.utils.**FormattedTextVerticalAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *Enum*

Vertical alignment of formatted text.

**BOTTOM = 'bottom'**

**MIDDLE = 'middle'**

**TOP = 'top'**

euporie.core.ft.utils.**add_border**(*ft: StyleAndTextTuples*, *width: int | None = None*, *style: str = ''*, *border_grid: GridStyle = ▯▯▯▯ ▯ ▯▯ ▯▯▯▯ ▯▯▯▯*, *border_visibility: DiBool | bool = True*, *border_style: DiStr | str = ''*, *padding: DiInt | int = 0*, *padding_style: DiStr | str = ''*) → StyleAndTextTuples

Add a border around formatted text.

**Parameters**

- **ft** – The formatted text to enclose with a border
- **width** – The target width including the border and padding
- **style** – The style to apply to the content background
- **border_grid** – The grid style to use for the border
- **border_visibility** – Determines which edges should receive a border
- **border_style** – The style to apply to the border
- **padding** – The width of spacing to apply between the content and the border
- **padding_style** – The style to apply to the border

**Returns**

The indented formatted text

euporie.core.ft.utils.**align**(*ft: StyleAndTextTuples*, *how: FormattedTextAlign = FormattedTextAlign.LEFT*, *width: int | None = None*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Align formatted text at a given width.

**Parameters**

- **how** – The alignment direction
- **ft** – The formatted text to strip
- **width** – The width to which the output should be padded. If `None`, the length of the longest line is used
- **style** – The style to apply to the padding
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – If True, whitespace will be ignored

**Returns**

The aligned formatted text

euporie.core.ft.utils.**apply_reverse_overwrites**(*ft: StyleAndTextTuples*) → StyleAndTextTuples

Write fragments tagged with "[ReverseOverwrite]" over text to their left.

euporie.core.ft.utils.**apply_style**(*ft: StyleAndTextTuples*, *style: str*) → StyleAndTextTuples

Apply a style to formatted text.

euporie.core.ft.utils.**concat**(*ft_a: StyleAndTextTuples*, *ft_b: StyleAndTextTuples*, *baseline_a: int = 0*, *baseline_b: int = 0*, *style: str = ''*) → tuple[StyleAndTextTuples, int]

Concatenate two blocks of formatted text, aligning at a given baseline.

**Parameters**

- **ft_a** – The first block of formatted text to combine
- **ft_b** – The second block of formatted text to combine
- **baseline_a** – The row to use to align the first block of formatted text with the second, counted in lines down from the top of the block
- **baseline_b** – The row to use to align the second block of formatted text with the second, counted in lines down from the top of the block
- **style** – The style to use for any extra lines added

**Returns**
A tuple containing the combined formatted text and the new baseline position

euporie.core.ft.utils.**fragment_list_to_words**(*fragments: StyleAndTextTuples*, *sep: str = ' '*) → Iterable[StyleAndTextTuples]

Split formatted text into a list of word fragments which form words.

euporie.core.ft.utils.**fragment_list_width**(*fragments: StyleAndTextTuples*) → int

Return the character width of this text fragment list.

Takes double width characters into account, and ignore special fragments: * ZeroWidthEscape * ReverseOverwrite

**Parameters**
**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

**Returns**
The width of the fragment list

euporie.core.ft.utils.**indent**(*ft: StyleAndTextTuples*, *margin: str = ' '*, *style: str = ''*, *skip_first: bool = False*) → StyleAndTextTuples

Indent formatted text with a given margin.

**Parameters**

- **ft** – The formatted text to strip
- **margin** – The margin string to add
- **style** – The style to apply to the margin
- **skip_first** – If `True`, the first line is skipped

**Returns**
The indented formatted text

euporie.core.ft.utils.**join_lines**(*fragments: list[StyleAndTextTuples]*) → StyleAndTextTuples

Join a list of lines of formatted text.

euporie.core.ft.utils.**last_char**(*ft: StyleAndTextTuples*) → str | None

Retrieve the last character of formatted text.

euporie.core.ft.utils.**lex**(*ft: StyleAndTextTuples*, *lexer_name: str*) → StyleAndTextTuples

Format formatted text using a named `pygments` lexer.

euporie.core.ft.utils.**max_line_width**(*ft: StyleAndTextTuples*) → int

Calculate the length of the longest line in formatted text.

euporie.core.ft.utils.**pad**(*ft: StyleAndTextTuples*, *width: int | None = None*, *char: str = ' '*, *style: str = ''*) → StyleAndTextTuples

Fill space at the end of lines.

euporie.core.ft.utils.**paste**(*ft_top: StyleAndTextTuples*, *ft_bottom: StyleAndTextTuples*, *row: int = 0*, *col: int = 0*, *transparent: bool = False*) → StyleAndTextTuples

Pate formatted text on top of other formatted text.

euporie.core.ft.utils.**strip**(*ft: StyleAndTextTuples*, *left: bool = True*, *right: bool = True*, *chars: str | None = None*, *only_unstyled: bool = False*) → StyleAndTextTuples

Strip whitespace (or a given character) from the ends of formatted text.

**Parameters**

- **ft** – The formatted text to strip
- **left** – If `True`, strip from the left side of the input
- **right** – If `True`, strip from the right side of the input
- **chars** – The character to strip. If `None`, strips whitespace
- **only_unstyled** – If `True`, only strip unstyled fragments

**Returns**

The stripped formatted text

euporie.core.ft.utils.**strip_one_trailing_newline**(*ft: StyleAndTextTuples*) → StyleAndTextTuples

Remove up to one trailing new-line character from formatted text.

euporie.core.ft.utils.**substring**(*ft: StyleAndTextTuples*, *start: int | None = None*, *end: int | None = None*) → StyleAndTextTuples

Extract a substring from formatted text.

euporie.core.ft.utils.**truncate**(*ft: StyleAndTextTuples*, *width: int*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Truncate all lines at a given length.

**Parameters**

- **ft** – The formatted text to truncate
- **width** – The width at which to truncate the text
- **style** – The style to apply to the truncation placeholder. The style of the truncated text will be used if not provided
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – Do not use placeholder when truncating whitespace

**Returns**

The truncated formatted text

euporie.core.ft.utils.**valign**(*ft: StyleAndTextTuples*, *how: FormattedTextVerticalAlign = FormattedTextVerticalAlign.MIDDLE*, *height: int | None = None*, *style: str = ''*) → StyleAndTextTuples

Align formatted text vertically.

euporie.core.ft.utils.**wrap**(*ft: StyleAndTextTuples*, *width: int*, *style: str = ''*, *placeholder: str = '…'*, *left: int = 0*, *truncate_long_words: bool = True*, *strip_trailing_ws: bool = False*, *margin: str = ''*) → StyleAndTextTuples

Wrap formatted text at a given width.

If words are longer than the given line they will be truncated

**Parameters**

- **ft** – The formatted text to wrap
- **width** – The width at which to wrap the text
- **style** – The style to apply to the truncation placeholder
- **placeholder** – The string that will appear at the end of a truncated line
- **left** – The starting position within the first line
- **truncate_long_words** – If `True` words longer than a line will be truncated
- **strip_trailing_ws** – If `True`, trailing whitespace will be removed from the ends of lines
- **margin** – Text to use a margin for the continuation of wrapped lines

**Returns**

The wrapped formatted text

### euporie.core.graphics

Define controls for display of terminal graphics.

### Functions

| | |
|---|---|
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |
| *ceil*(x, /) | Return the ceiling of x as an Integral. |
| *find_route*(from_, to) | Find the shortest conversion path between two formats. |
| *get_app*() | Get the current active (running) Application. |
| *get_cwidth*(string) | Return width of a string. |
| *passthrough*(cmd) | Wrap an escape sequence for terminal passthrough. |
| *select_graphic_control*(format_) | Determine which graphic control to use. |
| *split_lines*(fragments) | Take a single list of (style_str, text) tuples and yield one such list for each line. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |

### euporie.core.graphics.abstractmethod

euporie.core.graphics.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):

…

### euporie.core.graphics.ceil

euporie.core.graphics.**ceil**(*x*, */*)

Return the ceiling of x as an Integral.

This is the smallest integer >= x.

### euporie.core.graphics.find_route

euporie.core.graphics.**find_route**(*from_: str*, *to: str*) → list | None

Find the shortest conversion path between two formats.

### euporie.core.graphics.get_app

euporie.core.graphics.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.graphics.get_cwidth

euporie.core.graphics.**get_cwidth**(*string: str*) → int

Return width of a string. Wrapper around `wcwidth`.

### euporie.core.graphics.passthrough

euporie.core.graphics.**passthrough**(*cmd: str*) → str

Wrap an escape sequence for terminal passthrough.

### euporie.core.graphics.select_graphic_control

euporie.core.graphics.**select_graphic_control**(*format_: str*) → type[*euporie.core.graphics.GraphicControl*] | None

Determine which graphic control to use.

### euporie.core.graphics.split_lines

euporie.core.graphics.**split_lines**(*fragments: Iterable[OneStyleAndTextTuple]*) → Iterable[StyleAndTextTuples]

Take a single list of (style_str, text) tuples and yield one such list for each line. Just like str.split, this will yield at least one item.

**Parameters**

**fragments** – Iterable of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.graphics.to_filter

euporie.core.graphics.**to_filter**(*bool_or_filter: Union[*Filter, *bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### euporie.core.graphics.to_formatted_text

euporie.core.graphics.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

### Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *BoundedWritePosition*(xpos, ypos, width, height) | A write position which also hold bounding box information. |
| *Char*([char, style]) | Represent a single character in a *Screen*. |
| *Condition*(func) | Turn any callable into a Filter. |
| *Datum*(data, *args, **kwargs) | Class for storing and converting display data. |
| *DiInt*([top, right, bottom, left]) | A tuple of four integers with directions. |
| *FastDictCache*(get_value[, size]) | Fast, lightweight cache which keeps at most *size* items. |
| *Float*(content[, top, right, bottom, left, ...]) | Float for use in a *FloatContainer*. |
| *GraphicControl*(datum[, scale, bbox]) | A base-class for display controls which render terminal graphics. |
| *GraphicProcessor*(control) | Class which loads and positions graphics references in a *UIContent*. |
| *GraphicWindow*(content, get_position[, filter]) | A window responsible for displaying terminal graphics content. |
| *ItermGraphicControl*(datum[, scale, bbox]) | A graphic control which displays images using iTerm's graphics protocol. |
| *KittyGraphicControl*(datum[, scale, bbox]) | A graphic control which displays images using Kitty's graphics protocol. |
| *MouseHandlers*() | Two dimensional raster of callbacks for mouse events. |
| *Point*(x, y) | |
| *SimpleCache*([maxsize]) | Very simple cache that discards the oldest item when the cache size is exceeded. |
| *SixelGraphicControl*(datum[, scale, bbox]) | A graphic control which displays images as sixels. |
| *UIContent*(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| *UIControl*() | Base class for all user interface controls. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *WritePosition*(xpos, ypos, width, height) | |

### euporie.core.graphics.ABCMeta

**class** euporie.core.graphics.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.graphics.BoundedWritePosition

**class** euporie.core.graphics.**BoundedWritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*, *bbox:* euporie.core.data_structures.DiInt | *None = None*)

A write position which also hold bounding box information.

### euporie.core.graphics.Char

**class** euporie.core.graphics.**Char**(*char: str = ' '*, *style: str = ''*)

Represent a single character in a *Screen*.

This should be considered immutable.

**Parameters**

- **char** – A single character (can be a double-width character).
- **style** – A style string. (Can contain classnames.)

### euporie.core.graphics.Condition

**class** euporie.core.graphics.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.graphics.Datum

**class** euporie.core.graphics.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Class for storing and converting display data.

### euporie.core.graphics.DiInt

**class** euporie.core.graphics.**DiInt**(*top: int = 0*, *right: int = 0*, *bottom: int = 0*, *left: int = 0*)

A tuple of four integers with directions.

### euporie.core.graphics.FastDictCache

**class** euporie.core.graphics.**FastDictCache**(*get_value: Callable[[...], _V]*, *size: int = 1000000*)

Fast, lightweight cache which keeps at most *size* items. It will discard the oldest items in the cache first.

The cache is a dictionary, which doesn't keep track of access counts. It is perfect to cache little immutable objects which are not expensive to create, but where a dictionary lookup is still much faster than an object instantiation.

**Parameters**

**get_value** – Callable that's called in case of a missing key.

### euporie.core.graphics.Float

**class** euporie.core.graphics.**Float**(*content: AnyContainer*, *top: int | None = None*, *right: int | None = None*, *bottom: int | None = None*, *left: int | None = None*, *width: int | Callable[[], int] | None = None*, *height: int | Callable[[], int] | None = None*, *xcursor: bool = False*, *ycursor: bool = False*, *attach_to_window: AnyContainer | None = None*, *hide_when_covering_content: bool = False*, *allow_cover_cursor: bool = False*, *z_index: int = 1*, *transparent: bool = False*)

Float for use in a *FloatContainer*. Except for the *content* parameter, all other options are optional.

**Parameters**

- **content** – *Container* instance.
- **width** – *Dimension* or callable which returns a *Dimension*.
- **height** – *Dimension* or callable which returns a *Dimension*.
- **left** – Distance to the left edge of the *FloatContainer*.
- **right** – Distance to the right edge of the *FloatContainer*.
- **top** – Distance to the top of the *FloatContainer*.
- **bottom** – Distance to the bottom of the *FloatContainer*.
- **attach_to_window** – Attach to the cursor from this window, instead of the current window.
- **hide_when_covering_content** – Hide the float when it covers content underneath.
- **allow_cover_cursor** – When *False*, make sure to display the float below the cursor. Not on top of the indicated position.
- **z_index** – Z-index position. For a Float, this needs to be at least one. It is relative to the z_index of the parent container.
- **transparent** – *Filter* indicating whether this float needs to be drawn transparently.

### euporie.core.graphics.GraphicControl

**class** euporie.core.graphics.**GraphicControl**(*datum:* Datum, *scale: float = 0*, *bbox:* euporie.core.data_structures.DiInt | *None = None*)

A base-class for display controls which render terminal graphics.

### euporie.core.graphics.GraphicProcessor

**class** euporie.core.graphics.**GraphicProcessor**(*control:* UIControl)

Class which loads and positions graphics references in a `UIContent`.

### euporie.core.graphics.GraphicWindow

**class** euporie.core.graphics.**GraphicWindow**(*content:* GraphicControl, *get_position: Callable[[*Screen*],* BoundedWritePosition*]*, *filter: FilterOrBool = True*, **args: Any*, ***kwargs: Any*)

A window responsible for displaying terminal graphics content.

The content is displayed floating on top of a target window.

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.core.graphics.ItermGraphicControl

**class** euporie.core.graphics.**ItermGraphicControl**(*datum:* Datum, *scale: float = 0*, *bbox:* euporie.core.data_structures.DiInt | *None = None*)

A graphic control which displays images using iTerm's graphics protocol.

### euporie.core.graphics.KittyGraphicControl

**class** euporie.core.graphics.**KittyGraphicControl**(*datum:* Datum, *scale: float = 0*, *bbox:* euporie.core.data_structures.DiInt | *None = None*)

A graphic control which displays images using Kitty's graphics protocol.

### euporie.core.graphics.MouseHandlers

**class** euporie.core.graphics.**MouseHandlers**

Two dimensional raster of callbacks for mouse events.

### euporie.core.graphics.Point

**class** euporie.core.graphics.**Point**(*x*, *y*)

### euporie.core.graphics.SimpleCache

**class** euporie.core.graphics.**SimpleCache**(*maxsize: int = 8*)

Very simple cache that discards the oldest item when the cache size is exceeded.

**Parameters**

**maxsize** – Maximum size of the cache. (Don't make it too big.)

### euporie.core.graphics.SixelGraphicControl

**class** euporie.core.graphics.**SixelGraphicControl**(*datum:* Datum, *scale: float = 0*, *bbox:* euporie.core.data_structures.DiInt | *None = None*)

A graphic control which displays images as sixels.

### euporie.core.graphics.UIContent

**class** euporie.core.graphics.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>*, *line_count: int = 0*, *cursor_position: Point | None = None*, *menu_position: Point | None = None*, *show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.graphics.UIControl

**class** euporie.core.graphics.**UIControl**

Base class for all user interface controls.

### euporie.core.graphics.Window

**class** euporie.core.graphics.**Window**(*content:* UIControl | *None* = *None*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *dont_extend_width: FilterOrBool* = *False*, *dont_extend_height: FilterOrBool* = *False*, *ignore_content_width: FilterOrBool* = *False*, *ignore_content_height: FilterOrBool* = *False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool* = *False*, *wrap_lines: FilterOrBool* = *False*, *get_vertical_scroll: Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool* = *False*, *cursorline: FilterOrBool* = *False*, *cursorcolumn: FilterOrBool* = *False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]]* = *None*, *align:* WindowAlign | *Callable[[],* WindowAlign*]* = *WindowAlign.LEFT*, *style: str* | *Callable[[], str]* = *''*, *char: None* | *str* | *Callable[[], str]* = *None*, *get_line_prefix: GetLinePrefixCallable* | *None* = *None*)

Container that holds a control.

**Parameters**

- **content** – *UIControl* instance.
- **width** – *Dimension* instance or callable.
- **height** – *Dimension* instance or callable.
- **z_index** – When specified, this can be used to bring element in front of floating elements.
- **dont_extend_width** – When *True*, don't take up more width then the preferred width reported by the control.
- **dont_extend_height** – When *True*, don't take up more width then the preferred height reported by the control.
- **ignore_content_width** – A *bool* or *Filter* instance. Ignore the *UIContent* width when calculating the dimensions.
- **ignore_content_height** – A *bool* or *Filter* instance. Ignore the *UIContent* height when calculating the dimensions.
- **left_margins** – A list of *Margin* instance to be displayed on the left. For instance: NumberedMargin can be one of them in order to show line numbers.
- **right_margins** – Like *left_margins*, but on the other side.
- **scroll_offsets** – *ScrollOffsets* instance, representing the preferred amount of lines/columns to be always visible before/after the cursor. When both top and bottom are a very high number, the cursor will be centered vertically most of the time.
- **allow_scroll_beyond_bottom** – A *bool* or *Filter* instance. When True, allow scrolling so far, that the top part of the content is not visible anymore, while there is still empty space available at the bottom of the window. In the Vi editor for instance, this is possible. You will see tildes while the top part of the body is hidden.
- **wrap_lines** – A *bool* or *Filter* instance. When True, don't scroll horizontally, but wrap lines instead.

- **get_vertical_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll. (When this is *None*, the scroll is only determined by the last and current cursor position.)
- **get_horizontal_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll.
- **always_hide_cursor** – A *bool* or *Filter* instance. When True, never display the cursor, even when the user control specifies a cursor position.
- **cursorline** – A *bool* or *Filter* instance. When True, display a cursorline.
- **cursorcolumn** – A *bool* or *Filter* instance. When True, display a cursorcolumn.
- **colorcolumns** – A list of ColorColumn instances that describe the columns to be highlighted, or a callable that returns such a list.
- **align** – *WindowAlign* value or callable that returns an *WindowAlign* value. alignment of content.
- **style** – A style string. Style to be applied to all the cells in this window. (This can be a callable that returns a string.)
- **char** – (string) Character to be used for filling the background. This can also be a callable that returns a character.
- **get_line_prefix** – None or a callable that returns formatted text to be inserted before a line. It takes a line number (int) and a wrap_count and returns formatted text. This can be used for implementation of line continuations, things like Vim "breakindent" and so on.

### euporie.core.graphics.WritePosition

**class** euporie.core.graphics.**WritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*)

### Exceptions

| | |
|---|---|
| *NotVisible* | Exception to signal that a graphic is not currently visible. |

### euporie.core.graphics.NotVisible

**exception** euporie.core.graphics.**NotVisible**

Exception to signal that a graphic is not currently visible.

**class** euporie.core.graphics.**GraphicControl**(*datum:* Datum, *scale: float = 0*, *bbox:* euporie.core.data_structures.DiInt | *None = None*)

Bases: *UIControl*

A base-class for display controls which render terminal graphics.

**close**() → None

Remove the displayed object entirely.

**abstract convert_data**(*wp:* WritePosition) → str

Convert datum to required format.

**create_content**(*width: int*, *height: int*) → *UIContent*

Generate rendered output at a given size.

**Parameters**

- **width** – The desired output width
- **height** – The desired output height

**Returns**

*UIContent* for the given output size.

**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**() → *KeyBindingsBase* | None

The key bindings that are specific for this user control.

Return a `KeyBindings` object if some key bindings are specified, or *None* otherwise.

**abstract get_rendered_lines**(*width: int*, *height: int*, *wrap_lines: bool = False*) → list[StyleAndTextTuples]

Render the output data.

**hide**() → None

Hide the graphic from show.

**is_focusable**() → bool

Tell whether this user control is focusable.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

When *NotImplemented* is returned, it means that the given event is not handled by the *UIControl* itself. The *Window* or key bindings can decide to handle this event as scrolling or changing focus.

**Parameters**

**mouse_event** – *MouseEvent* instance.

**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: GetLinePrefixCallable* | *None*) → int

Return the number of lines in the rendered content.

**preferred_width**(*max_available_width: int*) → int | None

Return the width of the rendered content.

**reset**() → None

**class** euporie.core.graphics.**GraphicProcessor**(*control:* UIControl)

Bases: `object`

Class which loads and positions graphics references in a `UIContent`.

**get_graphic_float**(*key: str*) → *prompt_toolkit.layout.containers.Float* | None

Create a graphical float for an image.

**load**(*content:* UIContent) → None

Check for graphics in lines of text.

**class** euporie.core.graphics.**GraphicWindow**(*content:* GraphicControl, *get_position: Callable[[*Screen*]],* BoundedWritePosition*]*, *filter: FilterOrBool = True*, **args: Any*, ***kwargs: Any*)

Bases: *Window*

A window responsible for displaying terminal graphics content.

The content is displayed floating on top of a target window.

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: *GraphicControl***

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

Return the list of child *Container* objects.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Calculate the preferred height for this window.

**preferred_width**(*max_available_width: int*) → *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: int* | *None*) → None

Draw the graphic window's contents to the screen if required.

**class** euporie.core.graphics.**ItermGraphicControl**(*datum:* Datum, *scale: float = 0*, *bbox:* euporie.core.data_structures.DiInt | *None = None*)

Bases: *GraphicControl*

A graphic control which displays images using iTerm's graphics protocol.

**close**() → None

Remove the displayed object entirely.

**convert_data**(*wp:* WritePosition) → str

Convert the graphic's data to base64 data.

**create_content**(*width: int*, *height: int*) → *UIContent*

Generate rendered output at a given size.

**Parameters**

- **width** – The desired output width
- **height** – The desired output height

**Returns**

*UIContent* for the given output size.

**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**() → *KeyBindingsBase* | None

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*, *wrap_lines: bool* = *False*) → list[StyleAndTextTuples]

Get rendered lines from the cache, or generate them.

**hide**() → None

Hide the graphic from show.

**is_focusable**() → bool

Tell whether this user control is focusable.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

When *NotImplemented* is returned, it means that the given event is not handled by the *UIControl* itself. The *Window* or key bindings can decide to handle this event as scrolling or changing focus.

**Parameters**

**mouse_event** – *MouseEvent* instance.

**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:* *GetLinePrefixCallable* | *None*) → int

Return the number of lines in the rendered content.

**preferred_width**(*max_available_width: int*) → int | None

Return the width of the rendered content.

**reset**() → None

**class** euporie.core.graphics.**KittyGraphicControl**(*datum:* Datum, *scale: float* = *0*, *bbox:* euporie.core.data_structures.DiInt | *None* = *None*)

Bases: `GraphicControl`

A graphic control which displays images using Kitty's graphics protocol.

**close**() → None

Remove the displayed object entirely.

**convert_data**(*wp:* WritePosition) → str

Convert the graphic's data to base64 data for kitty graphics protocol.

**create_content**(*width: int*, *height: int*) → *UIContent*

Generate rendered output at a given size.

**Parameters**

- **width** – The desired output width
- **height** – The desired output height

**Returns**

*UIContent* for the given output size.

**delete**() → None

Delete the graphic from the terminal.

**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**() → *KeyBindingsBase* | None

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*, *wrap_lines: bool* = *False*) → list[StyleAndTextTuples]

Get rendered lines from the cache, or generate them.

**hide**() → None

Hide the graphic from show without deleting it.

**is_focusable**() → bool

Tell whether this user control is focusable.

**load**(*rows: int*, *cols: int*) → None

Send the graphic to the terminal without displaying it.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

When *NotImplemented* is returned, it means that the given event is not handled by the *UIControl* itself. The *Window* or key bindings can decide to handle this event as scrolling or changing focus.

**Parameters**

**mouse_event** – *MouseEvent* instance.

**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:* *GetLinePrefixCallable* | *None*) → int

Return the number of lines in the rendered content.

**preferred_width**(*max_available_width: int*) → int | None

Return the width of the rendered content.

**reset**() → None

Hide and delete the kitty graphic from the terminal.

**exception** euporie.core.graphics.**NotVisible**

Bases: Exception

Exception to signal that a graphic is not currently visible.

**add_note**()

Exception.add_note(note) – add a note to the exception

**args**

**with_traceback**()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

**class** euporie.core.graphics.**SixelGraphicControl**(*datum:* Datum, *scale: float = 0*, *bbox:* euporie.core.data_structures.DiInt | *None = None*)

Bases: *GraphicControl*

A graphic control which displays images as sixels.

**close**() → None

Remove the displayed object entirely.

**convert_data**(*wp:* WritePosition) → str

Convert datum to required format.

**create_content**(*width: int*, *height: int*) → *UIContent*

Generate rendered output at a given size.

**Parameters**

- **width** – The desired output width
- **height** – The desired output height

**Returns**

*UIContent* for the given output size.

**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**() → *KeyBindingsBase* | None

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*, *wrap_lines: bool = False*) → list[StyleAndTextTuples]

Get rendered lines from the cache, or generate them.

**hide**() → None

Hide the graphic from show.

**is_focusable**() → bool

Tell whether this user control is focusable.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

When *NotImplemented* is returned, it means that the given event is not handled by the *UIControl* itself. The *Window* or key bindings can decide to handle this event as scrolling or changing focus.

**Parameters**
**mouse_event** – *MouseEvent* instance.

**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: GetLinePrefixCallable | None*) → int

Return the number of lines in the rendered content.

**preferred_width**(*max_available_width: int*) → int | None

Return the width of the rendered content.

**reset**() → None

euporie.core.graphics.**select_graphic_control**(*format_: str*) → type[*euporie.core.graphics.GraphicControl*] | None

Determine which graphic control to use.

### euporie.core.history

Define input history loaders.

### Classes

| | |
|---|---|
| *History*() | Base `History` class. |
| *KernelHistory*(kernel[, n]) | Load the kernel's command history. |

### euporie.core.history.History

**class** euporie.core.history.**History**

Base `History` class.

This also includes abstract methods for loading/storing history.

### euporie.core.history.KernelHistory

**class** euporie.core.history.**KernelHistory**(*kernel:* Kernel, *n: int = 1000*)

Load the kernel's command history.

**class** euporie.core.history.**KernelHistory**(*kernel:* Kernel, *n: int = 1000*)

Bases: *History*

Load the kernel's command history.

**append_string**(*string: str*) → None

Add string to the history.

**get_strings**() → list[str]

Get the strings from the history that are loaded so far. (In order. Oldest item first.)

**async load**() → AsyncGenerator[str, None]

Load the history and yield all entries, most recent history first.

This method can be called multiple times from the *Buffer* to repopulate the history when prompting for a new input. So we are responsible here for both caching, and making sure that strings that were were appended to the history will be incorporated next time this method is called.

**Yields**
Each history string

**load_history_strings**() → Iterable[str]

Not used to load history, as we load it asynchronously.

**property recent: list[str]**

Return new items added since history was initially loaded.

**store_string**(*string: str*) → None

Don't store strings in persistent storage: they are stored by the kernel.

### euporie.core.inspection

Show contextual help for the current item under the cursor.

### Functions

| | |
|---|---|
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |

### euporie.core.inspection.abstractmethod

euporie.core.inspection.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):

…

## Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *FirstInspector*(inspectors) | Return results of the first inspector to response. |
| *Inspector*() | A class which provides contextual help on the item under the cursor. |
| *KernelInspector*(kernel) | Inspector which retrieves contextual help from a Jupyter kernel. |
| *LspInspector*(lsp, path) | Inspector which retrieves contextual help from a Language Server. |

### euporie.core.inspection.ABCMeta

**class** euporie.core.inspection.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.inspection.FirstInspector

**class** euporie.core.inspection.**FirstInspector**(*inspectors: Sequence[*Inspector*] | Callable[[], Sequence[*Inspector*]]*)

Return results of the first inspector to response.

### euporie.core.inspection.Inspector

**class** euporie.core.inspection.**Inspector**

A class which provides contextual help on the item under the cursor.

### euporie.core.inspection.KernelInspector

**class** euporie.core.inspection.**KernelInspector**(*kernel:* Kernel)

Inspector which retrieves contextual help from a Jupyter kernel.

### euporie.core.inspection.LspInspector

**class** euporie.core.inspection.**LspInspector**(*lsp:* LspClient, *path:* Path)

Inspector which retrieves contextual help from a Language Server.

**class** euporie.core.inspection.**FirstInspector**(*inspectors: Sequence[*Inspector*]* | *Callable[[], Sequence[*Inspector*]]*)

Bases: *Inspector*

Return results of the first inspector to response.

**async get_context**(*document:* Document, *auto:* *bool*) → dict[str, Any]

Request hover info from an LSP servers.

**class** euporie.core.inspection.**Inspector**

Bases: object

A class which provides contextual help on the item under the cursor.

**abstract async get_context**(*document:* Document, *auto:* *bool*) → dict[str, Any]

Call to retrieve the contextual help.

**class** euporie.core.inspection.**KernelInspector**(*kernel:* Kernel)

Bases: *Inspector*

Inspector which retrieves contextual help from a Jupyter kernel.

**async get_context**(*document:* Document, *auto:* *bool*) → dict[str, Any]

Request contextual help from the kernel.

**class** euporie.core.inspection.**LspInspector**(*lsp:* LspClient, *path:* Path)

Bases: *Inspector*

Inspector which retrieves contextual help from a Language Server.

**async get_context**(*document:* Document, *auto:* *bool*) → dict[str, Any]

Request hover info from an LSP servers.

### euporie.core.io

Define custom inputs and outputs, and related methods.

### Functions

| | |
|---|---|
| *b64encode*(s[, altchars]) | Encode the bytes-like object s using Base64 and return a bytes object. |

### euporie.core.io.b64encode

euporie.core.io.**b64encode**(*s*, *altchars=None*)

Encode the bytes-like object s using Base64 and return a bytes object.

Optional altchars should be a byte string of length 2 which specifies an alternative alphabet for the '+' and '/' characters. This allows an application to e.g. generate url or filesystem safe Base64 strings.

### Classes

| | |
|---|---|
| *DummyInput*() | Input for use in a *DummyApplication* |
| *IgnoredInput*() | An input which ignores input but does not immediately close the app. |
| *PseudoTTY*(underlying[, isatty]) | Make an output stream look like a TTY. |
| *PtkVt100_Output* | alias of *Vt100_Output* |
| *Vt100Parser*(*args, **kwargs) | A Vt100Parser which checks input against additional key patterns. |
| *Vt100_Output*(stdout, get_size[, term, ...]) | A Vt100 output which enables SGR pixel mouse positioning. |

### euporie.core.io.DummyInput

**class** euporie.core.io.**DummyInput**

Input for use in a *DummyApplication*

If used in an actual application, it will make the application render itself once and exit immediately, due to an *EOFError*.

### euporie.core.io.IgnoredInput

**class** euporie.core.io.**IgnoredInput**

An input which ignores input but does not immediately close the app.

### euporie.core.io.PseudoTTY

**class** euporie.core.io.**PseudoTTY**(*underlying: IO[str] | TextIO*, *isatty: bool = True*)

Make an output stream look like a TTY.

### euporie.core.io.PtkVt100_Output

euporie.core.io.**PtkVt100_Output**

alias of *Vt100_Output*

### euporie.core.io.Vt100Parser

**class** euporie.core.io.**Vt100Parser**(**args: Any*, ***kwargs: Any*)

A Vt100Parser which checks input against additional key patterns.

### euporie.core.io.Vt100_Output

**class** euporie.core.io.**Vt100_Output**(*stdout:* TextIO, *get_size: Callable[[],* Size*]*, *term: str* | *None* = *None*, *default_color_depth:* prompt_toolkit.output.color_depth.ColorDepth | *None* = *None*, *enable_bell: bool* = *True*, *enable_cpr: bool* = *True*)

A Vt100 output which enables SGR pixel mouse positioning.

**class** euporie.core.io.**IgnoredInput**

Bases: *DummyInput*

An input which ignores input but does not immediately close the app.

**attach**(*input_ready_callback: Callable[[], None]*) → ContextManager[None]

Do not call the callback, so the input is never closed.

**close**() → None

Close input.

**property closed: bool**

Should be true when the input stream is closed.

**cooked_mode**() → ContextManager[None]

Context manager that turns the input into cooked mode.

**detach**() → ContextManager[None]

Return a context manager that makes sure that this input is not active in the current event loop.

**fileno**() → int

Fileno for putting this in an event loop.

**flush**() → None

The event loop can call this when the input has to be flushed.

**flush_keys**() → list[prompt_toolkit.key_binding.key_processor.KeyPress]

Flush the underlying parser. and return the pending keys. (Used for vt100 input.)

**raw_mode**() → ContextManager[None]

Context manager that turns the input into raw mode.

**read_keys**() → list[prompt_toolkit.key_binding.key_processor.KeyPress]

Return a list of Key objects which are read/parsed from the input.

**typeahead_hash**() → str

Identifier for storing type ahead key presses.

**class** euporie.core.io.**PseudoTTY**(*underlying: IO[str]* | TextIO, *isatty: bool* = *True*)

Bases: object

Make an output stream look like a TTY.

**fake_tty = True**

**isatty**() → bool

Determine if the stream is interpreted as a TTY.

**class** euporie.core.io.**Vt100Parser**(**args: Any*, ***kwargs: Any*)

Bases: 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

**class** euporie.core.io.**Vt100_Output**(*stdout: TextIO*, *get_size: Callable[[], Size]*, *term: str | None = None*, *default_color_depth: prompt_toolkit.output.color_depth.ColorDepth | None = None*, *enable_bell: bool = True*, *enable_cpr: bool = True*)

Bases: *Vt100_Output*

A Vt100 output which enables SGR pixel mouse positioning.

**ask_for_cpr**() → None

Asks for a cursor position report (CPR).

**bell**() → None

Sound bell.

**clear_title**() → None

Clear title again. (or restore previous title.)

**cursor_backward**(*amount: int*) → None

Move cursor *amount* place backward.

**cursor_down**(*amount: int*) → None

Move cursor *amount* place down.

**cursor_forward**(*amount: int*) → None

Move cursor *amount* place forward.

**cursor_goto**(*row: int = 0*, *column: int = 0*) → None

Move cursor position.

**cursor_up**(*amount: int*) → None
Move cursor *amount* place up.

**disable_autowrap**() → None
Disable auto line wrapping.

**disable_bracketed_paste**() → None
For vt100 only.

**disable_extended_keys**() → None
Disable extended keys.

**disable_mouse_support**() → None
Additionally disable SGR-pixel mouse positioning.

**disable_private_sixel_colors**() → None
Disable private color registers for sixel graphics.

**enable_autowrap**() → None
Enable auto line wrapping.

**enable_bracketed_paste**() → None
For vt100 only.

**enable_extended_keys**() → None
Request extended keys.

**enable_mouse_support**() → None
Additionally enable SGR-pixel mouse positioning.

**enable_private_sixel_colors**() → None
Enable private color registers for sixel graphics.

**encoding**() → str
Return encoding used for stdout.

**enter_alternate_screen**() → None
Go to the alternate screen buffer. (For full screen applications).

**erase_down**() → None
Erases the screen from the current line down to the bottom of the screen.

**erase_end_of_line**() → None
Erases from the current cursor position to the end of the current line.

**erase_screen**() → None
Erases the screen with the background color and moves the cursor to home.

**fileno**() → int
Return file descriptor.

**flush**() → None
Write to output stream and flush.

**classmethod from_pty**(*stdout:* TextIO, *term: str | None = None*, *default_color_depth:* prompt_toolkit.output.color_depth.ColorDepth | *None = None*, *enable_bell: bool = True*) → *Vt100_Output*
Create an Output class from a pseudo terminal. (This will take the dimensions by reading the pseudo terminal attributes.)

**get_clipboard**() → None

Get clipboard contents using OSC-52.

**get_default_color_depth**() → *ColorDepth*

Return the default color depth for a vt100 terminal, according to the our term value.

We prefer 256 colors almost always, because this is what most terminals support these days, and is a good default.

**get_rows_below_cursor_position**() → int

For Windows only.

**get_size**() → *Size*

Return the size of the output window.

**hide_cursor**() → None

Hide cursor.

**quit_alternate_screen**() → None

Leave the alternate screen buffer.

**reset_attributes**() → None

Reset color and styling attributes.

**reset_cursor_key_mode**() → None

For vt100 only. Put the terminal in cursor mode (instead of application mode).

**reset_cursor_shape**() → None

Reset cursor shape.

**property responds_to_cpr: bool**

*True* if the *Application* can expect to receive a CPR response after calling *ask_for_cpr* (this will come back through the corresponding *Input*).

This is used to determine the amount of available rows we have below the cursor position. In the first place, we have this so that the drop down autocompletion menus are sized according to the available space.

On Windows, we don't need this, there we have *get_rows_below_cursor_position*.

**scroll_buffer_to_prompt**() → None

For Win32 only.

**set_attributes**(*attrs: Attrs*, *color_depth:* ColorDepth) → None

Create new style and output.

**Parameters**

**attrs** – *Attrs* instance.

**set_clipboard**(*text: str*) → None

Set clipboard data using OSC-52.

**set_cursor_shape**(*cursor_shape:* CursorShape) → None

Set cursor shape to block, beam or underline.

**set_title**(*title: str*) → None

Set terminal title.

**show_cursor**() → None

Show cursor.

**stdout**: *TextIO* = None

**write**(*data: str*) → None

Write text to output. (Removes vt100 escape codes. – used for safely writing text.)

**write_raw**(*data: str*) → None

Write raw data to output.

### euporie.core.kernel

Contain the main class for a notebook file.

### Functions

| | |
|---|---|
| *TypedDict*(typename[, fields, total]) | A simple typed namespace. |
| *jupyter_path*(*subdirs) | Return a list of directories to search for data files |
| *jupyter_runtime_dir*() | Return the runtime dir for transient jupyter files. |
| *uuid4*() | Generate a random UUID. |

### euporie.core.kernel.TypedDict

euporie.core.kernel.**TypedDict**(*typename*, *fields=None*, */*, *, *total=True*, ***kwargs*)

A simple typed namespace. At runtime it is equivalent to a plain dict.

TypedDict creates a dictionary type such that a type checker will expect all instances to have a certain set of keys, where each key is associated with a value of a consistent type. This expectation is not checked at runtime.

Usage:

```
class Point2D(TypedDict):
    x: int
    y: int
    label: str

a: Point2D = {'x': 1, 'y': 2, 'label': 'good'}  # OK
b: Point2D = {'z': 3, 'label': 'bad'}           # Fails type check

assert Point2D(x=1, y=2, label='first') == dict(x=1, y=2, label='first')
```

The type info can be accessed via the Point2D.__annotations__ dict, and the Point2D.__required_keys__ and Point2D.__optional_keys__ frozensets. TypedDict supports an additional equivalent form:

```
Point2D = TypedDict('Point2D', {'x': int, 'y': int, 'label': str})
```

By default, all keys must be present in a TypedDict. It is possible to override this by specifying totality:

```
class Point2D(TypedDict, total=False):
    x: int
    y: int
```

This means that a Point2D TypedDict can have any of the keys omitted. A type checker is only expected to support a literal False or True as the value of the total argument. True is the default, and makes all items defined in the class body be required.

The Required and NotRequired special forms can also be used to mark individual keys as being required or not required:

```
class Point2D(TypedDict):
    x: int               # the "x" key must always be present (Required is the␣
↪default)
    y: NotRequired[int]  # the "y" key can be omitted
```

See PEP 655 for more details on Required and NotRequired.

### euporie.core.kernel.jupyter_path

euporie.core.kernel.**jupyter_path**(**subdirs: str*) → list[str]

Return a list of directories to search for data files

JUPYTER_PATH environment variable has highest priority.

If the JUPYTER_PREFER_ENV_PATH environment variable is set, the environment-level directories will have priority over user-level directories.

If the Python site.ENABLE_USER_SITE variable is True, we also add the appropriate Python user site subdirectory to the user-level directories.

If `*subdirs` are given, that subdirectory will be added to each element.

Examples:

```
>>> jupyter_path()
['~/.local/jupyter', '/usr/local/share/jupyter']
>>> jupyter_path('kernels')
['~/.local/jupyter/kernels', '/usr/local/share/jupyter/kernels']
```

### euporie.core.kernel.jupyter_runtime_dir

euporie.core.kernel.**jupyter_runtime_dir**() → str

Return the runtime dir for transient jupyter files.

Returns JUPYTER_RUNTIME_DIR if defined.

The default is now (data_dir)/runtime on all platforms; we no longer use XDG_RUNTIME_DIR after various problems.

### euporie.core.kernel.uuid4

euporie.core.kernel.**uuid4**()

Generate a random UUID.

### Classes

| | |
|---|---|
| *AsyncKernelManager*(**kwargs) | An async kernel manager. |
| *EuporieKernelManager*(**kwargs) | Kernel Manager subclass. |
| *KPF* | alias of KernelProvisionerFactory |
| *Kernel*(kernel_tab[, threaded, allow_stdin, ...]) | Run a notebook kernel and communicates with it asynchronously. |
| *KernelManager*(**kwargs) | Manages a single kernel in a subprocess on this host. |
| *LocalProvisioner*(**kwargs) | *LocalProvisioner* is a concrete class of ABC KernelProvisionerBase and is the out-of-box default implementation used when no kernel provisioner is specified in the kernel specification (kernel.json). |
| *LoggingLocalProvisioner*(**kwargs) | A Jupyter kernel provisionser which logs kernel output. |
| *MsgCallbacks* | Typed dictionary for named message callbacks. |
| *UPath*(*args[, protocol]) | |
| *defaultdict* | defaultdict(default_factory=None, /, [...]) --> dict with default factory |

### euporie.core.kernel.AsyncKernelManager

**class** euporie.core.kernel.**AsyncKernelManager**(***kwargs: Any*)

An async kernel manager.

### euporie.core.kernel.EuporieKernelManager

**class** euporie.core.kernel.**EuporieKernelManager**(***kwargs: Any*)

Kernel Manager subclass.

jupyter_client replaces a plain python command with the current executable, but this is not desirable if the client is running in its own prefix (e.g. with pipx). We work around this here.

See https://github.com/jupyter/jupyter_client/issues/949

### euporie.core.kernel.KPF

euporie.core.kernel.**KPF**

alias of KernelProvisionerFactory

### euporie.core.kernel.Kernel

**class** euporie.core.kernel.**Kernel**(*kernel_tab:* KernelTab, *threaded: bool = True*, *allow_stdin: bool = False*, *default_callbacks:* MsgCallbacks | *None = None*, *connection_file:* Path | *None = None*)

Run a notebook kernel and communicates with it asynchronously.

Has the ability to run itself in it's own thread.

### euporie.core.kernel.KernelManager

**class** euporie.core.kernel.**KernelManager**(***kwargs: Any*)

Manages a single kernel in a subprocess on this host.

This version starts kernels with Popen.

### euporie.core.kernel.LocalProvisioner

**class** euporie.core.kernel.**LocalProvisioner**(***kwargs: Any*)

*LocalProvisioner* is a concrete class of ABC KernelProvisionerBase and is the out-of-box default implementation used when no kernel provisioner is specified in the kernel specification (kernel.json). It provides functional parity to existing applications by launching the kernel locally and using subprocess.Popen to manage its lifecycle.

This class is intended to be subclassed for customizing local kernel environments and serve as a reference implementation for other custom provisioners.

### euporie.core.kernel.LoggingLocalProvisioner

**class** euporie.core.kernel.**LoggingLocalProvisioner**(***kwargs: Any*)

A Jupyter kernel provisionser which logs kernel output.

### euporie.core.kernel.MsgCallbacks

**class** euporie.core.kernel.**MsgCallbacks**

Typed dictionary for named message callbacks.

### euporie.core.kernel.UPath

**class** euporie.core.kernel.**UPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

### euporie.core.kernel.defaultdict

**class** euporie.core.kernel.**defaultdict**

defaultdict(default_factory=None, /, […]) –> dict with default factory

The default factory is called without arguments to produce a new value when a key is not present, in __getitem__ only. A defaultdict compares equal to a dict with the same items. All remaining arguments are treated the same as if they were passed to the dict constructor, including keyword arguments.

### Exceptions

| | |
|---|---|
| *NoSuchKernel*(name) | An error raised when there is no kernel of a give name. |

### euporie.core.kernel.NoSuchKernel

**exception** euporie.core.kernel.**NoSuchKernel**(*name: str*)

An error raised when there is no kernel of a give name.

**class** euporie.core.kernel.**EuporieKernelManager**(***kwargs: Any*)

Bases: *AsyncKernelManager*

Kernel Manager subclass.

`jupyter_client` replaces a plain `python` command with the current executable, but this is not desirable if the client is running in its own prefix (e.g. with `pipx`). We work around this here.

See https://github.com/jupyter/jupyter_client/issues/949

**add_restart_callback**(*callback: Callable*, *event: str = 'restart'*) → None

Register a callback to be called when a kernel is restarted

**add_traits**(***traits: Any*) → None

Dynamically add trait attributes to the HasTraits instance.

**autorestart: Bool**

Should we autorestart the kernel if it dies.

**blocking_class**

A trait whose value must be a subclass of a specified class.

**blocking_client**() → BlockingKernelClient

Make a blocking client connected to my kernel

**cache_ports: Bool**

True if the MultiKernelManager should cache ports for this KernelManager instance

**classmethod class_config_rst_doc**() → str

Generate rST documentation for this class' config options.

Excludes traits defined on parent classes.

**classmethod class_config_section**(*classes: Optional[Sequence[type[traitlets.traitlets.HasTraits]]] = None*) → str

Get the config section for this class.

**Parameters**

**classes** (*list, optional*) – The list of other classes in the config file. Used to reduce redundant information.

**classmethod class_get_help**(*inst: traitlets.traitlets.HasTraits | None = None*) → str

Get the help string for this class in ReST format.

If *inst* is given, its current trait values will be used in place of class defaults.

**classmethod class_get_trait_help**(*trait: TraitType[Any, Any]*, *inst: traitlets.traitlets.HasTraits | None = None*, *helptext: str | None = None*) → str

Get the helptext string for a single trait.

**Parameters**

- **inst** – If given, its current trait values will be used in place of the class default.
- **helptext** – If not given, uses the *help* attribute of the current trait.

**classmethod class_own_trait_events**(*name: str*) → dict[str, traitlets.traitlets.EventHandler]

Get a dict of all event handlers defined on this class, not a parent.

Works like `event_handlers`, except for excluding traits from parents.

**classmethod class_own_traits**(***metadata: Any*) → dict[str, traitlets.traitlets.TraitType[Any, Any]]

Get a dict of all the traitlets defined on this class, not a parent.

Works like *class_traits*, except for excluding traits from parents.

**classmethod class_print_help**(*inst: traitlets.traitlets.HasTraits | None = None*) → None

Get the help string for a single trait and print it.

**classmethod class_trait_names**(***metadata: Any*) → list[str]

Get a list of all the names of this class' traits.

This method is just like the *trait_names()* method, but is unbound.

**classmethod class_traits**(***metadata: Any*) → dict[str, traitlets.traitlets.TraitType[Any, Any]]

Get a `dict` of all the traits of this class. The dictionary is keyed on the name and the values are the TraitType objects.

This method is just like the *traits()* method, but is unbound.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

**cleanup_connection_file**() → None

Cleanup connection file *if we wrote it*

Will not raise if the connection file was already removed somehow.

**cleanup_ipc_files**() → None

Cleanup ipc files if we wrote them.

**cleanup_random_ports**() → None

Forgets randomly assigned port numbers and cleans up the connection file.

Does nothing if no port numbers have been randomly assigned. In particular, does nothing unless the transport is tcp.

**async cleanup_resources**(*restart: bool = False*) → None

Clean up resources when the kernel is shut down

**client**(***kwargs: Any*) → AsyncKernelClient

Get a client for the manager.

**client_class: DottedObjectName**

A string holding a valid dotted object name in Python, such as A.b3._c

**client_factory: Type**

A trait whose value must be a subclass of a specified class.

**config**

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

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

Subclasses can declare default classes by overriding the klass attribute

**connect_control**(*identity: bytes | None = None*) → Socket

return zmq Socket connected to the Control channel

**connect_hb**(*identity: bytes | None = None*) → Socket

return zmq Socket connected to the Heartbeat channel

**connect_iopub**(*identity: bytes | None = None*) → Socket

return zmq Socket connected to the IOPub channel

**connect_shell**(*identity: bytes | None = None*) → Socket

return zmq Socket connected to the Shell channel

**connect_stdin**(*identity: bytes | None = None*) → Socket

return zmq Socket connected to the StdIn channel

**connection_file**

kernel-<pid>.json]

This file will contain the IP, ports, and authentication key needed to connect clients to this kernel. By default, this file will be created in the security dir of the current profile, but can be specified by absolute path.

**Type**

JSON file in which to store connection info [default

**context: Instance**

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

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

Subclasses can declare default classes by overriding the klass attribute

**control_port**

random]

**Type**

set the control (ROUTER) port [default

**property cross_validation_lock: Any**

A contextmanager for running a block with our cross validation lock set to True.

At the end of the block, the lock's value is restored to its value prior to entering the block.

**data_dir: str | Unicode**

A trait for unicode strings.

**async finish_shutdown**(*waittime: Optional[float] = None*, *pollinterval: float = 0.1*, *restart: bool = False*) → None

Wait for kernel shutdown, then kill process if it doesn't shutdown.

This does not send shutdown requests - use `request_shutdown()` first.

**format_kernel_cmd**(*extra_arguments: list[str] | None = None*) → list[str]

Replace templated args (e.g. {connection_file}).

**get_connection_info**(*session: bool = False*) → Dict[str, Union[int, str, bytes]]

Return the connection info as a dict

**Parameters**

**session** (*bool [default: False]*) – If True, return our session object will be included in the connection info. If False (default), the configuration parameters of our session object will be included, rather than the session object itself.

**Returns**

**connect_info** – dictionary of connection information.

**Return type**

dict

**property has_kernel: bool**

Has a kernel process been started that we are actively managing.

**has_trait**(*name: str*) → bool

Returns True if the object has a trait with the specified name.

**hb_port**

random]

**Type**

set the heartbeat port [default

**hold_trait_notifications**() → Any

Context manager for bundling trait change notifications and cross validation.

Use this when doing multiple trait assignments (init, config), to avoid race conditions in trait notifiers requesting other trait values. All trait notifications will fire after all values have been assigned.

**async interrupt_kernel**() → None

Interrupts the kernel by sending it a signal.

Unlike `signal_kernel`, this operation is well supported on all platforms.

**iopub_port**

random]

**Type**

set the iopub (PUB) port [default

**ip**

Set the kernel's IP address [default localhost]. If the IP address is something other than localhost, then Consoles on other machines will be able to connect to the Kernel, so be careful!

**property ipykernel: bool**

**async is_alive**() → bool

Is the kernel process still running?

**kernel_id: t.Union[str, Unicode]**

A trait for unicode strings.

**kernel_name: t.Union[str, Unicode]**

A trait for unicode strings.

**property kernel_spec: Optional[KernelSpec]**

**kernel_spec_manager: Instance**

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

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

Subclasses can declare default classes by overriding the klass attribute

**load_connection_file**(*connection_file: str | None = None*) → None

Load connection info from JSON dict in self.connection_file.

**Parameters**

**connection_file** (*unicode, optional*) – Path to connection file to load. If unspecified, use self.connection_file

**load_connection_info**(*info: Dict[str, Union[int, str, bytes]]*) → None

Load connection info from a dict containing connection info.

Typically this data comes from a connection file and is called by load_connection_file.

**Parameters**

**info** (*dict*) – Dictionary containing connection_info. See the connection_file spec for details.

**log**

Logger or LoggerAdapter instance

**notify_change**(*change: Bunch*) → None

Notify observers of a change event

**observe**(*handler: Callable[[...], Any]*, *names: Union[Sentinel, str, Iterable[traitlets.utils.sentinel.Sentinel | str]] = traitlets.All*, *type: traitlets.utils.sentinel.Sentinel | str = 'change'*) → None

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

**Parameters**

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)`, where `change` is a dictionary. The change dictionary at least holds a 'type' key. * `type`: the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

**on_trait_change**(*handler: traitlets.traitlets.EventHandler | None = None*, *name: traitlets.utils.sentinel.Sentinel | str | None = None*, *remove: bool = False*) → None

DEPRECATED: Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Static handlers can be created by creating methods on a HasTraits subclass with the naming convention '_[traitname]_changed'. Thus, to create static handler for the trait 'a', create the method _a_changed(self, name, old, new) (fewer arguments can be used, see below).

If *remove* is True and *handler* is not specified, all change handlers for the specified name are uninstalled.

**Parameters**

- **handler** (*callable, None*) – A callable that is called when a trait changes. Its signature can be handler(), handler(name), handler(name, new), handler(name, old, new), or handler(name, old, new, self).
- **name** (*list, str, None*) – If None, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **remove** (*bool*) – If False (the default), then install the handler. If True then unintall it.

**property owns_kernel: bool**

**parent**

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

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

Subclasses can declare default classes by overriding the klass attribute

**property ports: list[int]**

**async post_start_kernel**(***kw: Any*) → None

Performs any post startup tasks relative to the kernel.

**Parameters**

****kw** (*optional*) – keyword arguments that were used in the kernel process's launch.

**async pre_start_kernel**(***kw: Any*) → Tuple[List[str], Dict[str, Any]]

Prepares a kernel for startup in a separate process.

If random ports (port=0) are being used, this method must be called before the channels are created.

**Parameters**

****kw** (*optional*) – keyword arguments that are passed down to build the kernel_cmd and launching the kernel (e.g. Popen kwargs).

**provisioner: t.Optional[KernelProvisionerBase] = None**

**property ready: Union[Future, Future]**

A future that resolves when the kernel process has started for the first time

**remove_restart_callback**(*callback: Callable*, *event: str = 'restart'*) → None

Unregister a callback to be called when a kernel is restarted

**async request_shutdown**(*restart: bool = False*) → None

Send a shutdown request via control channel

**async restart_kernel**(*now: bool = False*, *newports: bool = False*, ***kw: Any*) → None

Restarts a kernel with the arguments that were used to launch it.

**Parameters**

- **now** (*bool, optional*) – If True, the kernel is forcefully restarted *immediately*, without having a chance to do any cleanup action. Otherwise the kernel is given 1s to clean up before a forceful restart is issued.

  In all cases the kernel is restarted, the only difference is whether it is given a chance to perform a clean shutdown or not.

- **newports** (*bool, optional*) – If the old kernel was launched with random ports, this flag decides whether the same ports and connection file will be used again. If False, the same ports and connection file are used. This is the default. If True, new random port numbers are chosen and a new connection file is written. It is still possible that the newly chosen random port numbers happen to be the same as the old ones.

- ****kw** (*optional*) – Any options specified here will overwrite those used to launch the kernel.

**classmethod section_names**() → list[str]

return section names as a list

**session**

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

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

Subclasses can declare default classes by overriding the klass attribute

**set_trait**(*name: str*, *value: Any*) → None

Forcibly sets trait attribute, including read-only attributes.

**setup_instance**(***kwargs: Any*) → None

This is called **before** self.__init__ is called.

**shell_port**

random]

**Type**
set the shell (ROUTER) port [default

**async shutdown_kernel**(*now: bool = False*, *restart: bool = False*) → None

Attempts to stop the kernel process cleanly.

This attempts to shutdown the kernels cleanly by:

1. Sending it a shutdown message over the control channel.
2. If that fails, the kernel is shutdown forcibly by sending it a signal.

**Parameters**

- **now** (*bool*) – Should the kernel be forcible killed *now*. This skips the first, nice shutdown attempt.

- **restart** (*bool*) – Will this kernel be restarted after it is shutdown. When this is True, connection files will not be cleaned up.

**shutdown_wait_time**: *Float*

Time to wait for a kernel to terminate before killing it, in seconds. When a shutdown request is initiated, the kernel will be immediately sent an interrupt (SIGINT), followedby a shutdown_request message, after 1/2 of *shutdown_wait_time`it will be sent a terminate (SIGTERM) request, and finally at the end of `shutdown_wait_time* will be killed (SIGKILL). terminate and kill may be equivalent on windows. Note that this value can beoverridden by the in-use kernel provisioner since shutdown times mayvary by provisioned environment.

**shutting_down**: bool = False

**async signal_kernel**(*signum: int*) → None

Sends a signal to the process group of the kernel (this usually includes the kernel and any subprocesses spawned by the kernel).

Note that since only SIGTERM is supported on Windows, this function is only useful on Unix systems.

**async start_kernel**(***kw: Any*) → None

Starts a kernel on this host in a separate process.

If random ports (port=0) are being used, this method must be called before the channels are created.

**Parameters**

****kw** (*optional*) – keyword arguments that are passed down to build the kernel_cmd and launching the kernel (e.g. Popen kwargs).

**start_restarter**() → None

Start the kernel restarter.

**stdin_port**

random]

**Type**

set the stdin (ROUTER) port [default

**stop_restarter**() → None

Stop the kernel restarter.

**trait_defaults**(**names: str*, ***metadata: Any*) → dict[str, Any] | traitlets.utils.sentinel.Sentinel

Return a trait's default value or a dictionary of them

### Notes

Dynamically generated default values may depend on the current state of the object.

**classmethod trait_events**(*name: str | None = None*) → dict[str, traitlets.traitlets.EventHandler]

Get a `dict` of all the event handlers of this class.

**Parameters**

**name** (*str (default: None)*) – The name of a trait of this class. If name is `None` then all the event handlers of this class will be returned instead.

**Return type**

The event handlers associated with a trait name, or all event handlers.

**trait_has_value**(*name: str*) → bool

Returns True if the specified trait has a value.

This will return false even if `getattr` would return a dynamically generated default value. These default values will be recognized as existing only after they have been generated.

Example

```
class MyClass(HasTraits):
    i = Int()


mc = MyClass()
assert not mc.trait_has_value("i")
mc.i  # generates a default value
assert mc.trait_has_value("i")
```

**trait_metadata** (*traitname: str*, *key: str*, *default: Any = None*) → Any

Get metadata values for trait by key.

**trait_names** (**metadata: Any*) → list[str]

Get a list of all the names of this class' traits.

**trait_values** (**metadata: Any*) → dict[str, Any]

A `dict` of trait names and their values.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

**Return type**
A `dict` of trait names and their values.

### Notes

Trait values are retrieved via `getattr`, any exceptions raised by traits or the operations they may trigger will result in the absence of a trait value in the result `dict`.

**traits** (**metadata: Any*) → dict[str, traitlets.traitlets.TraitType[Any, Any]]

Get a `dict` of all the traits of this class. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

**transport**

An enum of strings where the case should be ignored.

**unobserve** (*handler: Callable[[...], Any]*, *names: Union[Sentinel, str, Iterable[traitlets.utils.sentinel.Sentinel | str]] = traitlets.All*, *type: traitlets.utils.sentinel.Sentinel | str = 'change'*) → None

Remove a trait change handler.

This is used to unregister handlers to trait change notifications.

**Parameters**

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.

- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

**unobserve_all** (*name: str | Any = traitlets.All*) → None

Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

**update_config** (*config: Config*) → None

Update config and load the new values

**update_env** (*, *env: Dict[str, str]*) → None

Allow to update the environment of a kernel manager.

This will take effect only after kernel restart when the new env is passed to the new kernel.

This is useful as some of the information of the current kernel reflect the state of the session that started it, and those session information (like the attach file path, or name), are mutable.

**write_connection_file** (**kwargs: Any*) → None

Write connection info to JSON dict in self.connection_file.

**class** euporie.core.kernel.**Kernel** (*kernel_tab: KernelTab, threaded: bool = True, allow_stdin: bool = False, default_callbacks: MsgCallbacks | None = None, connection_file: Path | None = None*)

Bases: object

Run a notebook kernel and communicates with it asynchronously.

Has the ability to run itself in it's own thread.

**change** (*name: str | None, connection_file: Path | None = None, cb: Callable | None = None*) → None

Change the kernel.

**Parameters**

- **name** – The name of the kernel to change to
- **connection_file** – The path to the connection file to use
- **cb** – Callback to run once restarted

**comm_info** (*target_name: str | None = None*) → None

Request information about the current comms.

**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 stating 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, timeout: int = 60*) → list[dict]

Request code completions from the kernel, asynchronously.

**history**(*pattern: str = ''*, *n: int = 1*, *hist_access_type: str = 'search'*) → list[tuple[int, int, str]] | None

Retrieve history from the kernel.

**Parameters**

- **pattern** – The pattern to search for
- **n** – the number of history items to return
- **hist_access_type** – How to access the history ('range', 'tail' or 'search')

**Returns**

A list of history items, consisting of tuples (session, line_number, input)

**async history_**(*pattern: str = ''*, *n: int = 1*, *hist_access_type: str = 'search'*, *timeout: int = 1*) → list[tuple[int, int, str]] | None

Retrieve history from the kernel asynchronously.

**property id: str | None**

Get the ID of the current kernel.

**info**(*set_kernel_info: Callable[[dict[str, Any]], None] | None = None*, *set_status: Callable[[str], None] | None = None*) → None

Request information about the kernel.

**inspect**(*code: str*, *cursor_pos: int*, *callback: Callable[[dict[str, Any]], None] | 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*, *timeout: int = 2*) → 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.

**is_complete**(*code: str*, *timeout: int | float = 0.1*, *wait: bool = False*, *callback: Callable[[dict[str, Any]], None] | None = None*) → dict[str, Any]

Request code completeness status from the kernel.

**Parameters**

- **code** – The code string to check the completeness status of
- **timeout** – How long to wait for a kernel response
- **wait** – Whether to wait for the response
- **callback** – A function to run when the inspection result arrives. The result is passed as an argument.

**Returns**
A string describing the completeness status

**async is_complete_**(*code: str*, *timeout: int | float = 0.1*) → dict[str, Any]
Ask the kernel to determine if code is complete asynchronously.

**kc_comm**(*comm_id: str*, *data: dict[str, Any]*) → str
Send a comm message on the shell channel.

**property missing: bool**
Return True if the requested kernel is not found.

**async monitor_status**() → None
Regularly monitor the kernel status.

**on_iopub_clear_output**(*rsp: dict[str, Any]*) → None
Call callbacks for an iopub clear output response.

**on_iopub_comm_close**(*rsp: dict[str, Any]*) → None
Call callbacks for an iopub comm close response.

**on_iopub_comm_msg**(*rsp: dict[str, Any]*) → None
Call callbacks for an iopub comm message response.

**on_iopub_comm_open**(*rsp: dict[str, Any]*) → None
Call callbacks for an comm open response.

**on_iopub_display_data**(*rsp: dict[str, Any]*) → None
Call callbacks for an iopub display data response.

**on_iopub_error**(*rsp: dict[str, dict[str, Any]]*) → None
Call callbacks for an iopub error response.

**on_iopub_execute_input**(*rsp: dict[str, Any]*) → None
Call callbacks for an iopub execute input response.

**on_iopub_execute_result**(*rsp: dict[str, Any]*) → None
Call callbacks for an iopub execute result response.

**on_iopub_status**(*rsp: dict[str, Any]*) → None
Call callbacks for an iopub status response.

**on_iopub_stream**(*rsp: dict[str, Any]*) → None
Call callbacks for an iopub stream response.

**on_iopub_update_display_data**(*rsp: dict[str, Any]*) → None
Call callbacks for an iopub update display data response.

**on_shell_complete_reply**(*rsp: dict[str, Any]*) → None
Call callbacks for a shell completion reply response.

**on_shell_execute_reply**(*rsp: dict[str, Any]*) → None
Call callbacks for a shell execute reply response.

**on_shell_history_reply**(*rsp: dict[str, Any]*) → None
Call callbacks for a shell history reply response.

**on_shell_inspect_reply**(*rsp: dict[str, Any]*) → None
Call callbacks for a shell inspection reply response.

**on_shell_is_complete_reply**(*rsp: dict[str, Any]*) → None

Call callbacks for a shell completeness reply response.

**on_shell_kernel_info_reply**(*rsp: dict[str, Any]*) → None

Call callbacks for a shell kernel info response.

**on_shell_status**(*rsp: dict[str, Any]*) → None

Call `set_execution_count` callback for a shell status response.

**on_stdin_input_request**(*rsp: dict[str, Any]*) → None

Call `get_input` callback for a stdin input request message.

**on_unhandled**(*channel: str*, *rsp: dict[str, Any]*) → None

Report unhandled messages to the debug log.

**async poll**(*channel: str*) → None

Poll for messages on a channel, and signal when they arrive.

**Parameters**

**channel** – The name of the channel to get messages from

**async post_start_**() → None

Wait for the kernel to become ready.

**restart**(*wait: bool = False*, *cb: Callable | None = None*) → None

Restart the current kernel.

**async restart_**() → None

Restart the kernel asyncchronously.

**run**(*source: str*, *wait: bool = False*, *callback: Callable[..., None] | None = None*, ***callbacks: Callable[..., Any]*) → None

Run a cell using the notebook kernel and process the responses.

**async run_**(*source: str*, *get_input: Callable[[str, bool], None] | None = None*, *set_execution_count: Callable[[int], None] | None = None*, *add_output: Callable[[dict[str, Any]], None] | None = None*, *clear_output: Callable[[bool], None] | None = None*, *done: Callable[[dict[str, Any]], None] | None = None*, *set_metadata: Callable[[tuple[str, ...], Any], None] | None = None*, *set_status: Callable[[str], None] | None = None*) → None

Run the code cell and and set the response callbacks, optionally waiting.

**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 `Kernel` 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]**

Return a list of available kernelspecs.

**start**(*cb: Callable | None = None*, *wait: bool = False*, *timeout: int = 10*) → None

Start 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_**() → None

Start the kernel asynchronously and set its status.

**property status: str**

Retrieve the current kernel status.

**Returns**
The kernel status

**stop**(*cb: Callable | None = None*, *wait: bool = False*) → None

Stop 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: Callable[[], Any] | None = None*) → None

Stop the kernel asynchronously.

**wait_for_status**(*status: str = 'idle'*) → None

Block until the kernel reaches a given status value.

**class** euporie.core.kernel.**LoggingLocalProvisioner**(***kwargs: Any*)

Bases: *LocalProvisioner*

A Jupyter kernel provisionser which logs kernel output.

**add_traits**(***traits: Any*) → None

Dynamically add trait attributes to the HasTraits instance.

**classmethod class_config_rst_doc**() → str

Generate rST documentation for this class' config options.

Excludes traits defined on parent classes.

**classmethod class_config_section**(*classes: Optional[Sequence[type[traitlets.traitlets.HasTraits]]] = None*) → str

Get the config section for this class.

**Parameters**
**classes** (*list, optional*) – The list of other classes in the config file. Used to reduce redundant information.

**classmethod class_get_help**(*inst: traitlets.traitlets.HasTraits | None = None*) → str

Get the help string for this class in ReST format.

If *inst* is given, its current trait values will be used in place of class defaults.

**classmethod class_get_trait_help**(*trait: TraitType[Any, Any]*, *inst: traitlets.traitlets.HasTraits | None = None*, *helptext: str | None = None*) → str

Get the helptext string for a single trait.

**Parameters**

- **inst** – If given, its current trait values will be used in place of the class default.
- **helptext** – If not given, uses the *help* attribute of the current trait.

**classmethod class_own_trait_events**(*name: str*) → dict[str, traitlets.traitlets.EventHandler]

Get a dict of all event handlers defined on this class, not a parent.

Works like `event_handlers`, except for excluding traits from parents.

**classmethod class_own_traits**(**metadata: Any*) → dict[str, traitlets.traitlets.TraitType[Any, Any]]

Get a dict of all the traitlets defined on this class, not a parent.

Works like *class_traits*, except for excluding traits from parents.

**classmethod class_print_help**(*inst: traitlets.traitlets.HasTraits | None = None*) → None

Get the help string for a single trait and print it.

**classmethod class_trait_names**(**metadata: Any*) → list[str]

Get a list of all the names of this class' traits.

This method is just like the *trait_names()* method, but is unbound.

**classmethod class_traits**(**metadata: Any*) → dict[str, traitlets.traitlets.TraitType[Any, Any]]

Get a `dict` of all the traits of this class. The dictionary is keyed on the name and the values are the TraitType objects.

This method is just like the *traits()* method, but is unbound.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

**async cleanup**(*restart: bool = False*) → None

Clean up the resources used by the provisioner and optionally restart.

**config**

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

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

Subclasses can declare default classes by overriding the klass attribute

**connection_info: KernelConnectionInfo = {}**

**property cross_validation_lock: Any**

A contextmanager for running a block with our cross validation lock set to True.

At the end of the block, the lock's value is restored to its value prior to entering the block.

**async get_provisioner_info**() → Dict

Captures the base information necessary for persistence relative to this instance.

**get_shutdown_wait_time**(*recommended: float = 5.0*) → float

Returns the time allowed for a complete shutdown. This may vary by provisioner.

This method is called from *KernelManager.finish_shutdown()* during the graceful phase of its kernel shutdown sequence.

The recommended value will typically be what is configured in the kernel manager.

**get_stable_start_time**(*recommended: float = 10.0*) → float

Returns the expected upper bound for a kernel (re-)start to complete. This may vary by provisioner.

The recommended value will typically be what is configured in the kernel restarter.

**property has_process: bool**

Returns true if this provisioner is currently managing a process.

This property is asserted to be True immediately following a call to the provisioner's `launch_kernel()` method.

**has_trait**(*name: str*) → bool

Returns True if the object has a trait with the specified name.

**hold_trait_notifications**() → Any

Context manager for bundling trait change notifications and cross validation.

Use this when doing multiple trait assignments (init, config), to avoid race conditions in trait notifiers requesting other trait values. All trait notifications will fire after all values have been assigned.

**ip = None**

**kernel_id: Union[str, Unicode]**

A trait for unicode strings.

**kernel_spec: Any**

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

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

Subclasses can declare default classes by overriding the klass attribute

**async kill**(*restart: bool = False*) → None

Kill the provisioner and optionally restart.

**async launch_kernel**(*cmd: list[str], **kwargs: Any*) → KernelConnectionInfo

Launch a kernel with a command.

**async load_provisioner_info**(*provisioner_info: Dict*) → None

Loads the base information necessary for persistence relative to this instance.

**log**

Logger or LoggerAdapter instance

**notify_change**(*change: Bunch*) → None

Notify observers of a change event

**observe**(*handler: Callable[[...], Any], names: Union[Sentinel, str, Iterable[traitlets.utils.sentinel.Sentinel | str]] = traitlets.All, type: traitlets.utils.sentinel.Sentinel | str = 'change'*) → None

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

**Parameters**

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)`, where `change` is a dictionary. The change dictionary at least holds a 'type' key. * `type`: the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute *

`new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.

- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

**on_trait_change**(*handler: traitlets.traitlets.EventHandler | None = None*, *name: traitlets.utils.sentinel.Sentinel | str | None = None*, *remove: bool = False*) → None

DEPRECATED: Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Static handlers can be created by creating methods on a HasTraits subclass with the naming convention '_[traitname]_changed'. Thus, to create static handler for the trait 'a', create the method _a_changed(self, name, old, new) (fewer arguments can be used, see below).

If *remove* is True and *handler* is not specified, all change handlers for the specified name are uninstalled.

**Parameters**

- **handler** (*callable, None*) – A callable that is called when a trait changes. Its signature can be handler(), handler(name), handler(name, new), handler(name, old, new), or handler(name, old, new, self).
- **name** (*list, str, None*) – If None, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **remove** (*bool*) – If False (the default), then install the handler. If True then unintall it.

**parent**

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

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

Subclasses can declare default classes by overriding the klass attribute

**pgid = None**

**pid = None**

**async poll**() → Optional[int]

Poll the provisioner.

**ports_cached = False**

**async post_launch**(***kwargs: Any*) → None

Perform any steps following the kernel process launch.

This method is called from *KernelManager.post_start_kernel()* as part of its start kernel sequence.

**async pre_launch**(***kwargs: Any*) → Dict[str, Any]

Perform any steps in preparation for kernel process launch.

This includes applying additional substitutions to the kernel launch command and env. It also includes preparation of launch parameters.

Returns the updated kwargs.

**process = None**

**classmethod section_names**() → list[str]

return section names as a list

**async send_signal**(*signum: int*) → None

Sends a signal to the process group of the kernel (this usually includes the kernel and any subprocesses spawned by the kernel).

Note that since only SIGTERM is supported on Windows, we will check if the desired signal is for interrupt and apply the applicable code on Windows in that case.

**set_trait**(*name: str*, *value: Any*) → None

Forcibly sets trait attribute, including read-only attributes.

**setup_instance**(***kwargs: Any*) → None

This is called **before** self.__init__ is called.

**async shutdown_requested**(*restart: bool = False*) → None

Allows the provisioner to determine if the kernel's shutdown has been requested.

This method is called from *KernelManager.request_shutdown()* as part of its shutdown sequence.

This method is optional and is primarily used in scenarios where the provisioner may need to perform other operations in preparation for a kernel's shutdown.

**async terminate**(*restart: bool = False*) → None

Terminate the provisioner and optionally restart.

**trait_defaults**(**names: str*, ***metadata: Any*) → dict[str, Any] | traitlets.utils.sentinel.Sentinel

Return a trait's default value or a dictionary of them

### Notes

Dynamically generated default values may depend on the current state of the object.

**classmethod trait_events**(*name: str | None = None*) → dict[str, traitlets.traitlets.EventHandler]

Get a `dict` of all the event handlers of this class.

**Parameters**

**name** (*`str` `(default: None)`*) – The name of a trait of this class. If name is `None` then all the event handlers of this class will be returned instead.

**Return type**

The event handlers associated with a trait name, or all event handlers.

**trait_has_value**(*name: str*) → bool

Returns True if the specified trait has a value.

This will return false even if `getattr` would return a dynamically generated default value. These default values will be recognized as existing only after they have been generated.

Example

```
class MyClass(HasTraits):
    i = Int()


mc = MyClass()
assert not mc.trait_has_value("i")
```

(continues on next page)

(continued from previous page)

```
mc.i  # generates a default value
assert mc.trait_has_value("i")
```

**trait_metadata** (*traitname: str*, *key: str*, *default: Any = None*) → Any

Get metadata values for trait by key.

**trait_names** (**metadata: Any*) → list[str]

Get a list of all the names of this class' traits.

**trait_values** (**metadata: Any*) → dict[str, Any]

A `dict` of trait names and their values.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

**Return type**
A `dict` of trait names and their values.

### Notes

Trait values are retrieved via `getattr`, any exceptions raised by traits or the operations they may trigger will result in the absence of a trait value in the result `dict`.

**traits** (**metadata: Any*) → dict[str, traitlets.traitlets.TraitType[Any, Any]]

Get a `dict` of all the traits of this class. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

**unobserve** (*handler: Callable[[...], Any]*, *names: Union[Sentinel, str, Iterable[traitlets.utils.sentinel.Sentinel | str]] = traitlets.All*, *type: traitlets.utils.sentinel.Sentinel | str = 'change'*) → None

Remove a trait change handler.

This is used to unregister handlers to trait change notifications.

**Parameters**

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

**unobserve_all** (*name: str | Any = traitlets.All*) → None

Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

**update_config**(*config: Config*) → None

Update config and load the new values

**async wait**() → Optional[int]

Wait for the provisioner process.

**class** euporie.core.kernel.**MsgCallbacks**

Bases: TypedDict

Typed dictionary for named message callbacks.

**add_output: Callable[[dict[str, Any]], None] | None**

**ask_exit: Callable[[bool], None] | None**

**clear**() → None. Remove all items from D.

**clear_output: Callable[[bool], None] | None**

**completeness_status: Callable[[dict[str, Any]], None] | None**

**copy**() → a shallow copy of D

**dead: Callable[[], None] | None**

**done: Callable[[dict[str, Any]], None] | None**

**edit_magic: Callable[[str, int], None] | None**

**fromkeys**(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

**get**(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

**get_input: Callable[[str, bool], None] | None**

**items**() → a set-like object providing a view on D's items

**keys**() → a set-like object providing a view on D's keys

**page: Callable[[list[dict], int], None] | None**

**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If the key is not found, return the default if given; otherwise, raise a KeyError.

**popitem**()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

**set_execution_count: Callable[[int], None] | None**

**set_kernel_info: Callable[[dict[str, Any]], None] | None**

**set_metadata: Callable[[tuple[str, ...], Any], None] | None**

**set_next_input: Callable[[str, bool], None] | None**

**set_status: Callable[[str], None] | None**

**setdefault**(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**update**(*[E ]*, ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values**() → an object providing a view on D's values

### euporie.core.key_binding

Define key-bindings for the application.

#### Modules

| | |
|---|---|
| *euporie.core.key_binding.bindings* | Define collections of generic key-bindings which do not belong to widgets. |
| *euporie.core.key_binding.key_processor* | Modify the KeyProcessor to remove any timeout after an escape key press. |
| *euporie.core.key_binding.micro_state* | Define the state of the micro editing mode. |
| *euporie.core.key_binding.registry* | Define default key-bindings. |
| *euporie.core.key_binding.utils* | Utility functions for formatting key-bindings. |
| *euporie.core.key_binding.vi_state* | Create a Vi state which defaults to navigation mode. |

### euporie.core.key_binding.bindings

Define collections of generic key-bindings which do not belong to widgets.

#### Modules

| | |
|---|---|
| *euporie.core.key_binding.bindings.basic* | Define basic key-bindings for entering text. |
| *euporie.core.key_binding.bindings.completion* | Define editor key-bindings and commands for input completions. |
| *euporie.core.key_binding.bindings.micro* | Define editor key-bindings and commands for the micro editing mode. |
| *euporie.core.key_binding.bindings.mouse* | Key bindings to deal with pixel mouse positioning. |
| *euporie.core.key_binding.bindings.page_navigation* | Define page navigation key-bindings for buffers. |

### euporie.core.key_binding.bindings.basic

Define basic key-bindings for entering text.

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *if_no_repeat*(event) | Return True when the previous event was delivered to another handler. |
| *load_basic_bindings*([config]) | Load basic key-bindings for text entry. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *type_key*(event) | Enter a key. |

### euporie.core.key_binding.bindings.basic.add_cmd

euporie.core.key_binding.bindings.basic.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.key_binding.bindings.basic.if_no_repeat

euporie.core.key_binding.bindings.basic.**if_no_repeat**(*event:* KeyPressEvent) → bool

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

### euporie.core.key_binding.bindings.basic.load_basic_bindings

euporie.core.key_binding.bindings.basic.**load_basic_bindings**(*config:* Config | *None* = *None*) → *KeyBindingsBase*

Load basic key-bindings for text entry.

### euporie.core.key_binding.bindings.basic.load_registered_bindings

euporie.core.key_binding.bindings.basic.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.core.key_binding.bindings.basic.register_bindings

euporie.core.key_binding.bindings.basic.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.key_binding.bindings.basic.type_key

euporie.core.key_binding.bindings.basic.**type_key**(*event:* KeyPressEvent) → None

Enter a key.

### Classes

| | |
|---|---|
| *ConditionalKeyBindings*(key_bindings[, filter]) | Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.::. |
| *TextEntry*() | Basic key-bindings for text entry. |

### euporie.core.key_binding.bindings.basic.ConditionalKeyBindings

**class** euporie.core.key_binding.bindings.basic.**ConditionalKeyBindings**(*key_bindings:* KeyBindingsBase, *filter: Union[*Filter, *bool] = True*)

Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.:

```
@Condition
def setting_is_true():
    return True  # or False

registry = ConditionalKeyBindings(key_bindings, setting_is_true)
```

When new key bindings are added to this object. They are also enable/disabled according to the given *filter*.

**Parameters**

- **registries** – List of *KeyBindings* objects.
- **filter** – Filter object.

### euporie.core.key_binding.bindings.basic.TextEntry

**class** euporie.core.key_binding.bindings.basic.**TextEntry**

Basic key-bindings for text entry.

**class** euporie.core.key_binding.bindings.basic.**TextEntry**

Bases: object

Basic key-bindings for text entry.

euporie.core.key_binding.bindings.basic.**load_basic_bindings**(*config:* Config | *None* = *None*) → *KeyBindingsBase*

Load basic key-bindings for text entry.

euporie.core.key_binding.bindings.basic.**type_key**(*event:* KeyPressEvent) → None

Enter a key.

### euporie.core.key_binding.bindings.completion

Define editor key-bindings and commands for input completions.

#### Functions

| | |
|---|---|
| *accept_completion*() | Accept a selected completion. |
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *cancel_completion*() | Cancel a completion. |
| *get_app*() | Get the current active (running) Application. |
| *menu_complete*(event) | Generate completions, or go to the next completion. |
| *menu_complete_backward*(event) | Move backward through the list of possible completions. |
| *register_bindings*(bindings) | Update the key-binding registry. |

### euporie.core.key_binding.bindings.completion.accept_completion

euporie.core.key_binding.bindings.completion.**accept_completion**() → None

Accept a selected completion.

### euporie.core.key_binding.bindings.completion.add_cmd

euporie.core.key_binding.bindings.completion.**add_cmd**(**kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.key_binding.bindings.completion.cancel_completion

euporie.core.key_binding.bindings.completion.**cancel_completion**() → None

Cancel a completion.

### euporie.core.key_binding.bindings.completion.get_app

euporie.core.key_binding.bindings.completion.**get_app**() → *Application*[Any]

Get the current active (running) Application. An *Application* is active during the Application.run_async() call.

We assume that there can only be one *Application* active at the same time. There is only one terminal window, with only one stdin and stdout. This makes the code significantly easier than passing around the *Application* everywhere.

If no *Application* is running, then return by default a DummyApplication. For practical reasons, we prefer to not raise an exception. This way, we don't have to check all over the place whether an actual *Application* was returned.

(For applications like pymux where we can have more than one *Application*, we'll use a work-around to handle that.)

### euporie.core.key_binding.bindings.completion.menu_complete

euporie.core.key_binding.bindings.completion.**menu_complete**(*event:* KeyPressEvent) → None

Generate completions, or go to the next completion. (This is the default way of completing input in prompt_toolkit.)

### euporie.core.key_binding.bindings.completion.menu_complete_backward

euporie.core.key_binding.bindings.completion.**menu_complete_backward**(*event:* KeyPressEvent) → None

Move backward through the list of possible completions.

### euporie.core.key_binding.bindings.completion.register_bindings

euporie.core.key_binding.bindings.completion.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### Classes

| | |
|---|---|
| *Completion*(text[, start_position, display, ...]) | **param text**<br>The new string that will be inserted into the document. |

### euporie.core.key_binding.bindings.completion.Completion

**class** euporie.core.key_binding.bindings.completion.**Completion**(*text: str*, *start_position: int = 0*, *display: AnyFormattedText | None = None*, *display_meta: AnyFormattedText | None = None*, *style: str = ''*, *selected_style: str = ''*)

**Parameters**

- **text** – The new string that will be inserted into the document.
- **start_position** – Position relative to the cursor_position where the new text will start. The text will be inserted between the start_position and the original cursor position.
- **display** – (optional string or formatted text) If the completion has to be displayed differently in the completion menu.
- **display_meta** – (Optional string or formatted text) Meta information about the completion, e.g. the path or source where it's coming from. This can also be a callable that returns a string.
- **style** – Style string.
- **selected_style** – Style string, used for a selected completion. This can override the *style* parameter.

euporie.core.key_binding.bindings.completion.**accept_completion**() → None

Accept a selected completion.

euporie.core.key_binding.bindings.completion.**cancel_completion**() → None

Cancel a completion.

### euporie.core.key_binding.bindings.micro

Define editor key-bindings and commands for the micro editing mode.

### Functions

| | |
|---|---|
| *accept_line*(event) | Accept the line regardless of where the cursor is. |
| *accept_suggestion*(event) | Accept suggestion. |
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *backward_delete_char*(event) | Delete the character behind the cursor. |
| *backward_kill_word*(event) | Delete the word behind the cursor, using whitespace as a word boundary. |
| *backward_word*(event) | Move back to the start of the current or previous word. |
| *beginning_of_buffer*(event) | Move to the start of the buffer. |
| *cancel_selection*(event) | Cancel the selection. |
| *copy_selection*() | Add the current selection to the clipboard. |
| *cut_line*() | Remove the current line adds it to the clipboard. |
| *cut_selection*() | Remove the current selection and adds it to the clipboard. |
| *delete_char*(event) | Delete character before the cursor. |
| *delete_selection*() | Delete the contents of the current selection. |
| *dent_buffer*(event[, indenting]) | Indent or unindent the current or selected lines in a buffer. |
| *duplicate_line*() | Duplicate the current line. |
| *duplicate_selection*() | Duplicate the current selection. |
| *end_macro*() | Stop recording a macro. |
| *end_of_buffer*(event) | Move to the end of the buffer. |
| *extend_enum*(enumeration, name, *args, **kwds) | Add a new member to an existing Enum. |
| *extend_selection*(event) | Extend the selection. |
| *fill_suggestion*(event) | Fill partial suggestion. |
| *forward_word*(event) | Move forward to the end of the next word. |
| *get_app*() | Get the current active (running) Application. |
| *get_by_name*(name) | Return the handler for the (Readline) command with the given name. |
| *get_cmd*(name) | Get a command from the centralized command system by name. |
| *go_to_end_of_line*() | Move the cursor to the end of the line. |
| *go_to_end_of_paragraph*() | Move the cursor to the end of the current paragraph. |
| *go_to_matching_bracket*(event) | Go to matching bracket if the cursor is on a paired bracket. |
| *go_to_start_of_line*() | Move the cursor to the start of the line. |
| *go_to_start_of_paragraph*() | Move the cursor to the start of the current paragraph. |
| *if_no_repeat*(event) | Return True when the previous event was delivered to another handler. |
| *indent*(buffer, from_row, to_row[, count]) | Indent text of a *Buffer* object. |
| *indent_lines*(event) | Inndent the current or selected lines. |
| *load_micro_bindings*([config]) | Load editor key-bindings in the style of the `micro` text editor. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *move_cursor_left*() | Move back a character, or up a line. |
| *move_cursor_right*() | Move forward a character, or down a line. |
| *move_line*(n) | Move the current or selected lines up or down by one or more lines. |
| *move_lines_down*() | Move the current or selected lines down by one line. |
| *move_lines_up*() | Move the current or selected lines up by one line. |
| *newline*(event) | Inert a new line, replacing any selection and indenting if appropriate. |

continues on next page

Table 7 – continued from previous page

| | |
|---|---|
| *paste_clipboard*() | Pate the clipboard contents, replacing any current selection. |
| *redo*() | Redo the last edit. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *replace_selection*(event) | Replace selection by what is typed. |
| *run_macro*() | Re-execute the last keyboard macro defined. |
| *scroll_backward*(event[, half]) | Scroll window up. |
| *scroll_forward*(event[, half]) | Scroll window down. |
| *scroll_half_page_down*(event) | Same as ControlF, but only scroll half a page. |
| *scroll_half_page_up*(event) | Same as ControlB, but only scroll half a page. |
| *scroll_one_line_down*(event) | scroll_offset += 1 |
| *scroll_one_line_up*(event) | scroll_offset -= 1 |
| *select_all*() | Select all text. |
| *start_macro*() | Start recording a macro. |
| *start_selection*(event) | Start a new selection. |
| *toggle_case*() | Toggle the case of the current word or selection. |
| *toggle_comment*() | Comment or uncomments the current or selected lines. |
| *toggle_overwrite_mode*() | Toggle overwrite when using micro editing mode. |
| *undo*() | Undo the last edit. |
| *unindent*(buffer, from_row, to_row[, count]) | Unindent text of a *Buffer* object. |
| *unindent_lines*(event) | Unindent the current or selected lines. |
| *unshift_move*(event) | Handle keys in shift selection mode. |
| *wrap_selection_cmd*(left, right) | Add strings to either end of the current selection. |

### euporie.core.key_binding.bindings.micro.accept_line

euporie.core.key_binding.bindings.micro.**accept_line**(*event:* KeyPressEvent) → None

Accept the line regardless of where the cursor is.

### euporie.core.key_binding.bindings.micro.accept_suggestion

euporie.core.key_binding.bindings.micro.**accept_suggestion**(*event:* KeyPressEvent) → None

Accept suggestion.

### euporie.core.key_binding.bindings.micro.add_cmd

euporie.core.key_binding.bindings.micro.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.key_binding.bindings.micro.backward_delete_char

euporie.core.key_binding.bindings.micro.**backward_delete_char**(*event:* KeyPressEvent) → None

Delete the character behind the cursor.

### euporie.core.key_binding.bindings.micro.backward_kill_word

euporie.core.key_binding.bindings.micro.**backward_kill_word**(*event:* KeyPressEvent) → None

Delete the word behind the cursor, using whitespace as a word boundary.

### euporie.core.key_binding.bindings.micro.backward_word

euporie.core.key_binding.bindings.micro.**backward_word**(*event:* KeyPressEvent) → None

Move back to the start of the current or previous word. Words are composed of letters and digits.

### euporie.core.key_binding.bindings.micro.beginning_of_buffer

euporie.core.key_binding.bindings.micro.**beginning_of_buffer**(*event:* KeyPressEvent) → None

Move to the start of the buffer.

### euporie.core.key_binding.bindings.micro.cancel_selection

euporie.core.key_binding.bindings.micro.**cancel_selection**(*event:* KeyPressEvent) → None

Cancel the selection.

### euporie.core.key_binding.bindings.micro.copy_selection

euporie.core.key_binding.bindings.micro.**copy_selection**() → None

Add the current selection to the clipboard.

### euporie.core.key_binding.bindings.micro.cut_line

euporie.core.key_binding.bindings.micro.**cut_line**() → None

Remove the current line adds it to the clipboard.

### euporie.core.key_binding.bindings.micro.cut_selection

euporie.core.key_binding.bindings.micro.**cut_selection**() → None

Remove the current selection and adds it to the clipboard.

### euporie.core.key_binding.bindings.micro.delete_char

euporie.core.key_binding.bindings.micro.**delete_char**(*event:* KeyPressEvent) → None

Delete character before the cursor.

### euporie.core.key_binding.bindings.micro.delete_selection

euporie.core.key_binding.bindings.micro.**delete_selection**() → None

Delete the contents of the current selection.

### euporie.core.key_binding.bindings.micro.dent_buffer

euporie.core.key_binding.bindings.micro.**dent_buffer**(*event:* KeyPressEvent, *indenting:* *bool* = *True*) → None

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

### euporie.core.key_binding.bindings.micro.duplicate_line

euporie.core.key_binding.bindings.micro.**duplicate_line**() → None

Duplicate the current line.

### euporie.core.key_binding.bindings.micro.duplicate_selection

euporie.core.key_binding.bindings.micro.**duplicate_selection**() → None

Duplicate the current selection.

### euporie.core.key_binding.bindings.micro.end_macro

euporie.core.key_binding.bindings.micro.**end_macro**() → None

Stop recording a macro.

### euporie.core.key_binding.bindings.micro.end_of_buffer

euporie.core.key_binding.bindings.micro.**end_of_buffer**(*event:* KeyPressEvent) → None
Move to the end of the buffer.

### euporie.core.key_binding.bindings.micro.extend_enum

euporie.core.key_binding.bindings.micro.**extend_enum**(*enumeration*, *name*, **args*, ***kwds*)
Add a new member to an existing Enum.

### euporie.core.key_binding.bindings.micro.extend_selection

euporie.core.key_binding.bindings.micro.**extend_selection**(*event:* KeyPressEvent) → None
Extend the selection.

### euporie.core.key_binding.bindings.micro.fill_suggestion

euporie.core.key_binding.bindings.micro.**fill_suggestion**(*event:* KeyPressEvent) → None
Fill partial suggestion.

### euporie.core.key_binding.bindings.micro.forward_word

euporie.core.key_binding.bindings.micro.**forward_word**(*event:* KeyPressEvent) → None
Move forward to the end of the next word. Words are composed of letters and digits.

### euporie.core.key_binding.bindings.micro.get_app

euporie.core.key_binding.bindings.micro.**get_app**() → *BaseApp*
Get the current active (running) Application.

### euporie.core.key_binding.bindings.micro.get_by_name

euporie.core.key_binding.bindings.micro.**get_by_name**(*name: str*) → *Binding*
Return the handler for the (Readline) command with the given name.

### euporie.core.key_binding.bindings.micro.get_cmd

euporie.core.key_binding.bindings.micro.**get_cmd**(*name: str*) → *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.core.key_binding.bindings.micro.go_to_end_of_line

euporie.core.key_binding.bindings.micro.**go_to_end_of_line**() → None
Move the cursor to the end of the line.

### euporie.core.key_binding.bindings.micro.go_to_end_of_paragraph

euporie.core.key_binding.bindings.micro.**go_to_end_of_paragraph**() → None
Move the cursor to the end of the current paragraph.

### euporie.core.key_binding.bindings.micro.go_to_matching_bracket

euporie.core.key_binding.bindings.micro.**go_to_matching_bracket**(*event:* KeyPressEvent) → None
Go to matching bracket if the cursor is on a paired bracket.

### euporie.core.key_binding.bindings.micro.go_to_start_of_line

euporie.core.key_binding.bindings.micro.**go_to_start_of_line**() → None
Move the cursor to the start of the line.

### euporie.core.key_binding.bindings.micro.go_to_start_of_paragraph

euporie.core.key_binding.bindings.micro.**go_to_start_of_paragraph**() → None
Move the cursor to the start of the current paragraph.

### euporie.core.key_binding.bindings.micro.if_no_repeat

euporie.core.key_binding.bindings.micro.**if_no_repeat**(*event:* KeyPressEvent) → bool
Return True when the previous event was delivered to another handler.

### euporie.core.key_binding.bindings.micro.indent

euporie.core.key_binding.bindings.micro.**indent**(*buffer:* Buffer, *from_row: int*, *to_row: int*, *count: int* = *1*) → None
Indent text of a *Buffer* object.

### euporie.core.key_binding.bindings.micro.indent_lines

euporie.core.key_binding.bindings.micro.**indent_lines**(*event:* KeyPressEvent) → None

Inndent the current or selected lines.

### euporie.core.key_binding.bindings.micro.load_micro_bindings

euporie.core.key_binding.bindings.micro.**load_micro_bindings**(*config:* Config | *None* = *None*) → *KeyBindingsBase*

Load editor key-bindings in the style of the `micro` text editor.

### euporie.core.key_binding.bindings.micro.load_registered_bindings

euporie.core.key_binding.bindings.micro.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.core.key_binding.bindings.micro.move_cursor_left

euporie.core.key_binding.bindings.micro.**move_cursor_left**() → None

Move back a character, or up a line.

### euporie.core.key_binding.bindings.micro.move_cursor_right

euporie.core.key_binding.bindings.micro.**move_cursor_right**() → None

Move forward a character, or down a line.

### euporie.core.key_binding.bindings.micro.move_line

euporie.core.key_binding.bindings.micro.**move_line**(*n: int*) → None

Move the current or selected lines up or down by one or more lines.

### euporie.core.key_binding.bindings.micro.move_lines_down

euporie.core.key_binding.bindings.micro.**move_lines_down**() → None

Move the current or selected lines down by one line.

### euporie.core.key_binding.bindings.micro.move_lines_up

euporie.core.key_binding.bindings.micro.**move_lines_up**() → None

Move the current or selected lines up by one line.

### euporie.core.key_binding.bindings.micro.newline

euporie.core.key_binding.bindings.micro.**newline**(*event:* KeyPressEvent) → None

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

### euporie.core.key_binding.bindings.micro.paste_clipboard

euporie.core.key_binding.bindings.micro.**paste_clipboard**() → None

Pate the clipboard contents, replacing any current selection.

### euporie.core.key_binding.bindings.micro.redo

euporie.core.key_binding.bindings.micro.**redo**() → None

Redo the last edit.

### euporie.core.key_binding.bindings.micro.register_bindings

euporie.core.key_binding.bindings.micro.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.key_binding.bindings.micro.replace_selection

euporie.core.key_binding.bindings.micro.**replace_selection**(*event:* KeyPressEvent) → None

Replace selection by what is typed.

### euporie.core.key_binding.bindings.micro.run_macro

euporie.core.key_binding.bindings.micro.**run_macro**() → None

Re-execute the last keyboard macro defined.

### euporie.core.key_binding.bindings.micro.scroll_backward

euporie.core.key_binding.bindings.micro.**scroll_backward**(*event:* KeyPressEvent, *half:* *bool* = *False*) → None

Scroll window up.

### euporie.core.key_binding.bindings.micro.scroll_forward

euporie.core.key_binding.bindings.micro.**scroll_forward**(*event:* KeyPressEvent, *half:* *bool* = *False*) → None

Scroll window down.

### euporie.core.key_binding.bindings.micro.scroll_half_page_down

euporie.core.key_binding.bindings.micro.**scroll_half_page_down**(*event:* KeyPressEvent) → None

Same as ControlF, but only scroll half a page.

### euporie.core.key_binding.bindings.micro.scroll_half_page_up

euporie.core.key_binding.bindings.micro.**scroll_half_page_up**(*event:* KeyPressEvent) → None

Same as ControlB, but only scroll half a page.

### euporie.core.key_binding.bindings.micro.scroll_one_line_down

euporie.core.key_binding.bindings.micro.**scroll_one_line_down**(*event:* KeyPressEvent) → None

scroll_offset += 1

### euporie.core.key_binding.bindings.micro.scroll_one_line_up

euporie.core.key_binding.bindings.micro.**scroll_one_line_up**(*event:* KeyPressEvent) → None

scroll_offset -= 1

### euporie.core.key_binding.bindings.micro.select_all

euporie.core.key_binding.bindings.micro.**select_all**() → None

Select all text.

### euporie.core.key_binding.bindings.micro.start_macro

euporie.core.key_binding.bindings.micro.**start_macro**() → None

Start recording a macro.

### euporie.core.key_binding.bindings.micro.start_selection

euporie.core.key_binding.bindings.micro.**start_selection**(*event:* KeyPressEvent) → None

Start a new selection.

### euporie.core.key_binding.bindings.micro.toggle_case

euporie.core.key_binding.bindings.micro.**toggle_case**() → None

Toggle the case of the current word or selection.

### euporie.core.key_binding.bindings.micro.toggle_comment

euporie.core.key_binding.bindings.micro.**toggle_comment**() → None

Comment or uncomments the current or selected lines.

### euporie.core.key_binding.bindings.micro.toggle_overwrite_mode

euporie.core.key_binding.bindings.micro.**toggle_overwrite_mode**() → None

Toggle overwrite when using micro editing mode.

### euporie.core.key_binding.bindings.micro.undo

euporie.core.key_binding.bindings.micro.**undo**() → None

Undo the last edit.

### euporie.core.key_binding.bindings.micro.unindent

euporie.core.key_binding.bindings.micro.**unindent**(*buffer:* Buffer, *from_row: int*, *to_row: int*, *count: int = 1*) → None

Unindent text of a *Buffer* object.

### euporie.core.key_binding.bindings.micro.unindent_lines

euporie.core.key_binding.bindings.micro.**unindent_lines**(*event:* KeyPressEvent) → None

Unindent the current or selected lines.

### euporie.core.key_binding.bindings.micro.unshift_move

euporie.core.key_binding.bindings.micro.**unshift_move**(*event:* KeyPressEvent) → None

Handle keys in 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.core.key_binding.bindings.micro.wrap_selection_cmd

euporie.core.key_binding.bindings.micro.**wrap_selection_cmd**(*left: str*, *right: str*) → None

Add strings to either end of the current selection.

## Classes

| | |
|---|---|
| *ConditionalKeyBindings*(key_bindings[, filter]) | Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.::. |
| *Document*([text, cursor_position, selection]) | This is a immutable class around the text and cursor position, and contains methods for querying this data, e.g. |
| *EditMode*() | Micro style editor key-bindings. |
| *EditingMode*(value[, names, module, ...]) | |
| *Keys*(value[, names, module, qualname, type, ...]) | List of keys for use in key bindings. |
| *MicroInputMode*(value[, names, module, ...]) | Enum to define edit mode state types. |
| *SelectionState*([original_cursor_position, type]) | State of the current selection. |
| *SelectionType*(value[, names, module, ...]) | Type of selection. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.key_binding.bindings.micro.ConditionalKeyBindings

**class** euporie.core.key_binding.bindings.micro.**ConditionalKeyBindings**(*key_bindings:* KeyBindingsBase, *filter:* *Union*[Filter, *bool*] = *True*)

Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.:

```
@Condition
def setting_is_true():
    return True  # or False
```

(continues on next page)

(continued from previous page)

```
registry = ConditionalKeyBindings(key_bindings, setting_is_true)
```

When new key bindings are added to this object. They are also enable/disabled according to the given *filter*.

**Parameters**

- **registries** – List of *KeyBindings* objects.
- **filter** – Filter object.

### euporie.core.key_binding.bindings.micro.Document

**class** euporie.core.key_binding.bindings.micro.**Document**(*text: str = ''*, *cursor_position: int | None = None*, *selection: prompt_toolkit.selection.SelectionState | None = None*)

This is a immutable class around the text and cursor position, and contains methods for querying this data, e.g. to give the text before the cursor.

This class is usually instantiated by a *Buffer* object, and accessed as the *document* property of that class.

**Parameters**

- **text** – string
- **cursor_position** – int
- **selection** – *SelectionState*

### euporie.core.key_binding.bindings.micro.EditMode

**class** euporie.core.key_binding.bindings.micro.**EditMode**

Micro style editor key-bindings.

### euporie.core.key_binding.bindings.micro.EditingMode

**class** euporie.core.key_binding.bindings.micro.**EditingMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.key_binding.bindings.micro.Keys

**class** euporie.core.key_binding.bindings.micro.**Keys**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

List of keys for use in key bindings.

Note that this is an "StrEnum", all values can be compared against strings.

### euporie.core.key_binding.bindings.micro.MicroInputMode

**class** euporie.core.key_binding.bindings.micro.**MicroInputMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Enum to define edit mode state types.

### euporie.core.key_binding.bindings.micro.SelectionState

**class** euporie.core.key_binding.bindings.micro.**SelectionState**(*original_cursor_position: int = 0*, *type: SelectionType = SelectionType.CHARACTERS*)

State of the current selection.

**Parameters**

- **original_cursor_position** – int
- **type** – *SelectionType*

### euporie.core.key_binding.bindings.micro.SelectionType

**class** euporie.core.key_binding.bindings.micro.**SelectionType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Type of selection.

### euporie.core.key_binding.bindings.micro.partial

**class** euporie.core.key_binding.bindings.micro.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.key_binding.bindings.micro.**EditMode**

Bases: object

Micro style editor key-bindings.

euporie.core.key_binding.bindings.micro.**accept_suggestion**(*event:* KeyPressEvent) → None

Accept suggestion.

euporie.core.key_binding.bindings.micro.**backward_kill_word**(*event:* KeyPressEvent) → None

Delete the word behind the cursor, using whitespace as a word boundary.

euporie.core.key_binding.bindings.micro.**cancel_selection**(*event:* KeyPressEvent) → None

Cancel the selection.

euporie.core.key_binding.bindings.micro.**copy_selection**() → None

Add the current selection to the clipboard.

euporie.core.key_binding.bindings.micro.**cut_line**() → None

Remove the current line adds it to the clipboard.

euporie.core.key_binding.bindings.micro.**cut_selection**() → None

Remove the current selection and adds it to the clipboard.

euporie.core.key_binding.bindings.micro.**delete_selection**() → None

Delete the contents of the current selection.

euporie.core.key_binding.bindings.micro.**dent_buffer**(*event:* KeyPressEvent, *indenting: bool* = *True*) → None

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

euporie.core.key_binding.bindings.micro.**duplicate_line**() → None

Duplicate the current line.

euporie.core.key_binding.bindings.micro.**duplicate_selection**() → None

Duplicate the current selection.

euporie.core.key_binding.bindings.micro.**end_macro**() → None

Stop recording a macro.

euporie.core.key_binding.bindings.micro.**extend_selection**(*event:* KeyPressEvent) → None

Extend the selection.

euporie.core.key_binding.bindings.micro.**fill_suggestion**(*event:* KeyPressEvent) → None

Fill partial suggestion.

euporie.core.key_binding.bindings.micro.**go_to_end_of_line**() → None

Move the cursor to the end of the line.

euporie.core.key_binding.bindings.micro.**go_to_end_of_paragraph**() → None

Move the cursor to the end of the current paragraph.

euporie.core.key_binding.bindings.micro.**go_to_matching_bracket**(*event:* KeyPressEvent) → None

Go to matching bracket if the cursor is on a paired bracket.

euporie.core.key_binding.bindings.micro.**go_to_start_of_line**() → None

Move the cursor to the start of the line.

euporie.core.key_binding.bindings.micro.**go_to_start_of_paragraph**() → None

Move the cursor to the start of the current paragraph.

euporie.core.key_binding.bindings.micro.**indent_lines**(*event:* KeyPressEvent) → None

Inndent the current or selected lines.

euporie.core.key_binding.bindings.micro.**load_micro_bindings**(*config:* Config | *None* = *None*) → *KeyBindingsBase*

Load editor key-bindings in the style of the `micro` text editor.

euporie.core.key_binding.bindings.micro.**move_cursor_left**() → None

Move back a character, or up a line.

euporie.core.key_binding.bindings.micro.**move_cursor_right**() → None

Move forward a character, or down a line.

euporie.core.key_binding.bindings.micro.**move_line**(*n: int*) → None

Move the current or selected lines up or down by one or more lines.

euporie.core.key_binding.bindings.micro.**move_lines_down**() → None

Move the current or selected lines down by one line.

euporie.core.key_binding.bindings.micro.**move_lines_up**() → None

Move the current or selected lines up by one line.

euporie.core.key_binding.bindings.micro.**newline**(*event:* KeyPressEvent) → None

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

euporie.core.key_binding.bindings.micro.**paste_clipboard**() → None

Pate the clipboard contents, replacing any current selection.

euporie.core.key_binding.bindings.micro.**redo**() → None

Redo the last edit.

euporie.core.key_binding.bindings.micro.**replace_selection**(*event:* KeyPressEvent) → None

Replace selection by what is typed.

euporie.core.key_binding.bindings.micro.**run_macro**() → None

Re-execute the last keyboard macro defined.

euporie.core.key_binding.bindings.micro.**select_all**() → None

Select all text.

euporie.core.key_binding.bindings.micro.**start_macro**() → None

Start recording a macro.

euporie.core.key_binding.bindings.micro.**start_selection**(*event:* KeyPressEvent) → None

Start a new selection.

euporie.core.key_binding.bindings.micro.**toggle_case**() → None
Toggle the case of the current word or selection.

euporie.core.key_binding.bindings.micro.**toggle_comment**() → None
Comment or uncomments the current or selected lines.

euporie.core.key_binding.bindings.micro.**toggle_overwrite_mode**() → None
Toggle overwrite when using micro editing mode.

euporie.core.key_binding.bindings.micro.**undo**() → None
Undo the last edit.

euporie.core.key_binding.bindings.micro.**unindent_lines**(*event:* KeyPressEvent) → None
Unindent the current or selected lines.

euporie.core.key_binding.bindings.micro.**unshift_move**(*event:* KeyPressEvent) → None
Handle keys in 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.core.key_binding.bindings.micro.**wrap_selection_cmd**(*left: str*, *right: str*) → None
Add strings to either end of the current selection.

### euporie.core.key_binding.bindings.mouse

Key bindings to deal with pixel mouse positioning.

### Functions

| | |
|---|---|
| *NamedTuple*(typename[, fields]) | Typed version of namedtuple. |
| *load_mouse_bindings*() | Additional key-bindings to deal with SGR-pixel mouse positioning. |
| *load_ptk_mouse_bindings*() | Key bindings, required for mouse support. |

### euporie.core.key_binding.bindings.mouse.NamedTuple

euporie.core.key_binding.bindings.mouse.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)
Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

### euporie.core.key_binding.bindings.mouse.load_mouse_bindings

euporie.core.key_binding.bindings.mouse.**load_mouse_bindings**() → *KeyBindings*

Additional key-bindings to deal with SGR-pixel mouse positioning.

### euporie.core.key_binding.bindings.mouse.load_ptk_mouse_bindings

euporie.core.key_binding.bindings.mouse.**load_ptk_mouse_bindings**() → *KeyBindings*

Key bindings, required for mouse support. (Mouse events enter through the key binding system.)

### Classes

| | |
|---|---|
| *BaseApp*([title, set_title, leave_graphics, ...]) | All euporie apps. |
| *FastDictCache*(get_value[, size]) | Fast, lightweight cache which keeps at most *size* items. |
| *KeyBindings*() | A container for a set of key bindings. |
| *Keys*(value[, names, module, qualname, type, ...]) | List of keys for use in key bindings. |
| *MouseButton*(value[, names, module, ...]) | |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, which also store relative position of the mouse event in a cell. |
| *MouseEventType*(value[, names, module, ...]) | |
| *MouseModifier*(value[, names, module, ...]) | |
| *Point*(x, y) | |
| *PtkMouseEvent* | alias of *MouseEvent* |
| *RelativePosition*(x, y) | Store the relative position or the mouse within a terminal cell. |

### euporie.core.key_binding.bindings.mouse.BaseApp

**class** euporie.core.key_binding.bindings.mouse.**BaseApp**(*title: str | None = None*, *set_title: bool = True*, *leave_graphics: FilterOrBool = True*, *extend_renderer_height: FilterOrBool = False*, *extend_renderer_width: FilterOrBool = False*, *enable_page_navigation_bindings: FilterOrBool | None = True*, ***kwargs: Any*)

All euporie apps.

The base euporie application class.

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

### euporie.core.key_binding.bindings.mouse.FastDictCache

**class** euporie.core.key_binding.bindings.mouse.**FastDictCache**(*get_value: Callable[[...], _V]*, *size: int = 1000000*)

Fast, lightweight cache which keeps at most *size* items. It will discard the oldest items in the cache first.

The cache is a dictionary, which doesn't keep track of access counts. It is perfect to cache little immutable objects which are not expensive to create, but where a dictionary lookup is still much faster than an object instantiation.

**Parameters**

**get_value** – Callable that's called in case of a missing key.

### euporie.core.key_binding.bindings.mouse.KeyBindings

**class** euporie.core.key_binding.bindings.mouse.**KeyBindings**

A container for a set of key bindings.

Example usage:

```
kb = KeyBindings()

@kb.add('c-t')
def _(event):
    print('Control-T pressed')

@kb.add('c-a', 'c-b')
def _(event):
    print('Control-A pressed, followed by Control-B')

@kb.add('c-x', filter=is_searching)
def _(event):
    print('Control-X pressed')  # Works only if we are searching.
```

### euporie.core.key_binding.bindings.mouse.Keys

**class** euporie.core.key_binding.bindings.mouse.**Keys**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

List of keys for use in key bindings.

Note that this is an "StrEnum", all values can be compared against strings.

### euporie.core.key_binding.bindings.mouse.MouseButton

**class** euporie.core.key_binding.bindings.mouse.**MouseButton**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.key_binding.bindings.mouse.MouseEvent

**class** euporie.core.key_binding.bindings.mouse.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers:* *frozenset*[prompt_toolkit.mouse_events.MouseModifier], *cell_position:* euporie.core.key_binding.bindings.mouse.RelativePosition | *None*)

Mouse event, which also store relative position of the mouse event in a cell.

### euporie.core.key_binding.bindings.mouse.MouseEventType

**class** euporie.core.key_binding.bindings.mouse.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.key_binding.bindings.mouse.MouseModifier

**class** euporie.core.key_binding.bindings.mouse.**MouseModifier**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.key_binding.bindings.mouse.Point

**class** euporie.core.key_binding.bindings.mouse.**Point**(*x*, *y*)

### euporie.core.key_binding.bindings.mouse.PtkMouseEvent

euporie.core.key_binding.bindings.mouse.**PtkMouseEvent**

alias of *MouseEvent*

### euporie.core.key_binding.bindings.mouse.RelativePosition

**class** euporie.core.key_binding.bindings.mouse.**RelativePosition**(*x: float*, *y: float*)

Store the relative position or the mouse within a terminal cell.

**class** euporie.core.key_binding.bindings.mouse.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers:* *frozenset*[prompt_toolkit.mouse_events.MouseModifier], *cell_position:* euporie.core.key_binding.bindings.mouse.RelativePosition | *None*)

Bases: *MouseEvent*

Mouse event, which also store relative position of the mouse event in a cell.

**class** euporie.core.key_binding.bindings.mouse.**RelativePosition**(*x: float*, *y: float*)

Bases: NamedTuple

Store the relative position or the mouse within a terminal cell.

**count**(*value*, */*)

Return number of occurrences of value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**x: float**

Alias for field number 0

**y: float**

Alias for field number 1

euporie.core.key_binding.bindings.mouse.**load_mouse_bindings**() → *KeyBindings*

Additional key-bindings to deal with SGR-pixel mouse positioning.

### euporie.core.key_binding.bindings.page_navigation

Define page navigation key-bindings for buffers.

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *get_app*() | Get the current active (running) Application. |
| *load_emacs_page_navigation_bindings*() | Key bindings, for scrolling up and down through pages. |
| *load_page_navigation_bindings*([config]) | Load page navigation key-bindings for text entry. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *load_vi_page_navigation_bindings*() | Key bindings, for scrolling up and down through pages. |
| *merge_key_bindings*(bindings) | Merge multiple Keybinding objects together. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *scroll_page_down*(event) | Scroll page down (prefer the cursor at the top of the page, after scrolling). |
| *scroll_page_up*(event) | Scroll page up (prefer the cursor at the bottom of the page, after scrolling). |

### euporie.core.key_binding.bindings.page_navigation.add_cmd

euporie.core.key_binding.bindings.page_navigation.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.key_binding.bindings.page_navigation.get_app

euporie.core.key_binding.bindings.page_navigation.**get_app**() → *Application*[Any]

Get the current active (running) Application. An *Application* is active during the `Application.run_async()` call.

We assume that there can only be one *Application* active at the same time. There is only one terminal window, with only one stdin and stdout. This makes the code significantly easier than passing around the *Application* everywhere.

If no *Application* is running, then return by default a `DummyApplication`. For practical reasons, we prefer to not raise an exception. This way, we don't have to check all over the place whether an actual *Application* was returned.

(For applications like pymux where we can have more than one *Application*, we'll use a work-around to handle that.)

### euporie.core.key_binding.bindings.page_navigation.load_emacs_page_navigation_bindings

euporie.core.key_binding.bindings.page_navigation.**load_emacs_page_navigation_bindings**() → *KeyBindingsBase*

Key bindings, for scrolling up and down through pages. This are separate bindings, because GNU readline doesn't have them.

### euporie.core.key_binding.bindings.page_navigation.load_page_navigation_bindings

euporie.core.key_binding.bindings.page_navigation.**load_page_navigation_bindings**(*config: Config | None = None*) → *KeyBindingsBase*

Load page navigation key-bindings for text entry.

### euporie.core.key_binding.bindings.page_navigation.load_registered_bindings

euporie.core.key_binding.bindings.page_navigation.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.core.key_binding.bindings.page_navigation.load_vi_page_navigation_bindings

euporie.core.key_binding.bindings.page_navigation.**load_vi_page_navigation_bindings**() → *KeyBindingsBase*

Key bindings, for scrolling up and down through pages. This are separate bindings, because GNU readline doesn't have them.

### euporie.core.key_binding.bindings.page_navigation.merge_key_bindings

euporie.core.key_binding.bindings.page_navigation.**merge_key_bindings**(*bindings: Sequence[*KeyBindingsBase*]*) → _MergedKeyBindings

Merge multiple `Keybinding` objects together.

Usage:

```
bindings = merge_key_bindings([bindings1, bindings2, ...])
```

### euporie.core.key_binding.bindings.page_navigation.register_bindings

euporie.core.key_binding.bindings.page_navigation.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.key_binding.bindings.page_navigation.scroll_page_down

euporie.core.key_binding.bindings.page_navigation.**scroll_page_down**(*event: KeyPressEvent*) → None

Scroll page down (prefer the cursor at the top of the page, after scrolling).

### euporie.core.key_binding.bindings.page_navigation.scroll_page_up

euporie.core.key_binding.bindings.page_navigation.**scroll_page_up**(*event: KeyPressEvent*) → None

Scroll page up (prefer the cursor at the bottom of the page, after scrolling).

### Classes

| | |
|---|---|
| *ConditionalKeyBindings*(key_bindings[, filter]) | Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.::. |
| *PageNavigation*() | Key-bindings for page navigation. |

### euporie.core.key_binding.bindings.page_navigation.ConditionalKeyBindings

**class** euporie.core.key_binding.bindings.page_navigation.**ConditionalKeyBindings**(*key_bindings: KeyBindingsBase, filter: Union[Filter, bool] = True*)

Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.:

```
@Condition
def setting_is_true():
    return True  # or False

registry = ConditionalKeyBindings(key_bindings, setting_is_true)
```

When new key bindings are added to this object. They are also enable/disabled according to the given *filter*.

**Parameters**

- **registries** – List of *KeyBindings* objects.
- **filter** – Filter object.

### euporie.core.key_binding.bindings.page_navigation.PageNavigation

**class** euporie.core.key_binding.bindings.page_navigation.**PageNavigation**

Key-bindings for page navigation.

**class** euporie.core.key_binding.bindings.page_navigation.**PageNavigation**

Bases: object

Key-bindings for page navigation.

euporie.core.key_binding.bindings.page_navigation.**load_page_navigation_bindings**(*config:* Config | *None* = *None*) → *KeyBindingsBase*

Load page navigation key-bindings for text entry.

euporie.core.key_binding.bindings.page_navigation.**scroll_page_down**(*event:* KeyPressEvent) → None

Scroll page down (prefer the cursor at the top of the page, after scrolling).

euporie.core.key_binding.bindings.page_navigation.**scroll_page_up**(*event:* KeyPressEvent) → None

Scroll page up (prefer the cursor at the bottom of the page, after scrolling).

### euporie.core.key_binding.key_processor

Modify the KeyProcessor to remove any timeout after an escape key press.

### Functions

| | |
|---|---|
| *get_app*() | Get the current active (running) Application. |

### euporie.core.key_binding.key_processor.get_app

euporie.core.key_binding.key_processor.**get_app**() → *Application*[Any]

Get the current active (running) Application. An *Application* is active during the `Application.run_async()` call.

We assume that there can only be one *Application* active at the same time. There is only one terminal window, with only one stdin and stdout. This makes the code significantly easier than passing around the *Application* everywhere.

If no *Application* is running, then return by default a `DummyApplication`. For practical reasons, we prefer to not raise an exception. This way, we don't have to check all over the place whether an actual *Application* was returned.

(For applications like pymux where we can have more than one *Application*, we'll use a work-around to handle that.)

### Classes

| | |
|---|---|
| *KeyProcessor*(*args, **kwargs) | A subclass of prompt_toolkit's keyprocessor. |
| *Keys*(value[, names, module, qualname, type, ...]) | List of keys for use in key bindings. |
| *PtKeyProcessor* | alias of *KeyProcessor* |

### euporie.core.key_binding.key_processor.KeyProcessor

**class** euporie.core.key_binding.key_processor.**KeyProcessor**(**args: Any*, ***kwargs: Any*)

A subclass of prompt_toolkit's keyprocessor.

This adds an exception to the auto-flush timeout so that the input is flushed immediately if the key pressed is the escape key.

### euporie.core.key_binding.key_processor.Keys

**class** euporie.core.key_binding.key_processor.**Keys**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

List of keys for use in key bindings.

Note that this is an "StrEnum", all values can be compared against strings.

### euporie.core.key_binding.key_processor.PtKeyProcessor

euporie.core.key_binding.key_processor.**PtKeyProcessor**

alias of *KeyProcessor*

**class** euporie.core.key_binding.key_processor.**KeyProcessor**(**args: Any*, ***kwargs: Any*)

Bases: *KeyProcessor*

A subclass of prompt_toolkit's keyprocessor.

This adds an exception to the auto-flush timeout so that the input is flushed immediately if the key pressed is the escape key.

**empty_queue**() → list[prompt_toolkit.key_binding.key_processor.KeyPress]

Empty the input queue. Return the unprocessed input.

**feed**(*key_press: KeyPress*, *first: bool = False*) → None

Add a new `KeyPress` to the input queue. (Don't forget to call *process_keys* in order to process the queue.)

**Parameters**

**first** – If true, insert before everything else.

**feed_multiple**(*key_presses: list[prompt_toolkit.key_binding.key_processor.KeyPress]*, *first: bool = False*) → None

**Parameters**

**first** – If true, insert before everything else.

**process_keys**() → None

Process all the keys in the input queue.

**reset**() → None

**send_sigint**() → None

Send SIGINT. Immediately call the SIGINT key handler.

### euporie.core.key_binding.micro_state

Define the state of the micro editing mode.

### Classes

| | |
|---|---|
| *Enum*(value[, names, module, qualname, type, ...]) | Create a collection of name/value pairs. |
| *MicroInputMode*(value[, names, module, ...]) | Enum to define edit mode state types. |
| *MicroState*() | Mutable class to hold Micro specific state. |

### euporie.core.key_binding.micro_state.Enum

**class** euporie.core.key_binding.micro_state.**Enum**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes – see the documentation for details.

### euporie.core.key_binding.micro_state.MicroInputMode

**class** euporie.core.key_binding.micro_state.**MicroInputMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Enum to define edit mode state types.

### euporie.core.key_binding.micro_state.MicroState

**class** euporie.core.key_binding.micro_state.**MicroState**

Mutable class to hold Micro specific state.

**class** euporie.core.key_binding.micro_state.**MicroInputMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: str, *Enum*

Enum to define edit mode state types.

**INSERT = 'insert'**

**REPLACE = 'replace'**

**capitalize**()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

**casefold**()

Return a version of the string suitable for caseless comparisons.

**center**(*width*, *fillchar=' '*, */*)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

**count**(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

**encode**(*encoding='utf-8'*, *errors='strict'*)

Encode the string using the codec registered for encoding.

**encoding**
The encoding in which to encode the string.

**errors**
The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

**endswith**(*suffix*[, *start*[, *end*]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

**expandtabs**(*tabsize=8*)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

**find**(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

**format**(**args*, ***kwargs*) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

**format_map**(*mapping*) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

**index**(*sub*[, *start*[, *end* ]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

**isalnum**()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

**isalpha**()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

**isascii**()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

**isdecimal**()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

**isdigit**()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

**isidentifier**()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as "def" or "class".

**islower**()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

**isnumeric**()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

**isprintable**()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

**isspace**()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

**istitle**()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

**isupper**()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

**join**(*iterable*, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

**ljust**(*width*, *fillchar=' '*, /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

**lower**()

Return a copy of the string converted to lowercase.

**lstrip**(*chars=None*, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

**static maketrans**()

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

**partition**(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

**removeprefix**(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return string[len(prefix):]. Otherwise, return a copy of the original string.

**removesuffix**(*suffix*, */*)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

**replace**(*old*, *new*, *count=-1*, */*)

Return a copy with all occurrences of substring old replaced by new.

**count**
Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

**rfind**(*sub*[, *start*[, *end* ]])  → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

**rindex**(*sub*[, *start*[, *end* ]])  → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

**rjust**(*width*, *fillchar=' '*, */*)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

**rpartition**(*sep*, */*)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

**rsplit**(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using sep as the separator string.

**sep**
The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

**maxsplit**
Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

**rstrip**(*chars=None*, */*)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

**split**(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using sep as the separator string.

**sep**
The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

**maxsplit**
Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

**splitlines**(*keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

**startswith**(*prefix*[, *start*[, *end*]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

**strip**(*chars=None*, */*)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

**swapcase**()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

**title**()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

**translate**(*table*, */*)

Replace each character in the string using the given translation table.

**table**
Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via __getitem__, for instance a dictionary or list. If this operation raises LookupError, the character is left untouched. Characters mapped to None are deleted.

**upper**()

Return a copy of the string converted to uppercase.

**value: str**

**zfill**(*width*, */*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

**class** euporie.core.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.core.key_binding.registry

Define default key-bindings.

### Functions

| | |
|---|---|
| *get_cmd*(name) | Get a command from the centralized command system by name. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *register_bindings*(bindings) | Update the key-binding registry. |

### euporie.core.key_binding.registry.get_cmd

euporie.core.key_binding.registry.**get_cmd**(*name: str*) → *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.core.key_binding.registry.load_registered_bindings

euporie.core.key_binding.registry.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*
Assign key-bindings to commands based on a dictionary.

### euporie.core.key_binding.registry.register_bindings

euporie.core.key_binding.registry.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### Classes

| | |
|---|---|
| *KeyBindings*() | A container for a set of key bindings. |

### euporie.core.key_binding.registry.KeyBindings

**class** euporie.core.key_binding.registry.**KeyBindings**

A container for a set of key bindings.

Example usage:

```
kb = KeyBindings()

@kb.add('c-t')
def _(event):
    print('Control-T pressed')

@kb.add('c-a', 'c-b')
def _(event):
    print('Control-A pressed, followed by Control-B')

@kb.add('c-x', filter=is_searching)
def _(event):
    print('Control-X pressed')  # Works only if we are searching.
```

euporie.core.key_binding.registry.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

euporie.core.key_binding.registry.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.key_binding.utils

Utility functions for formatting key-bindings.

### Functions

| | |
|---|---|
| *format_keys*(keys) | Convert a list of tuples of keys to a string. |
| *if_no_repeat*(event) | Return True when the previous event was delivered to another handler. |
| *parse_keys*(keys) | Pare a list of keys. |

#### euporie.core.key_binding.utils.format_keys

euporie.core.key_binding.utils.**format_keys**(*keys: list[tuple[str | prompt_toolkit.keys.Keys, ...]]*) → list[str]

Convert a list of tuples of keys to a string.

#### euporie.core.key_binding.utils.if_no_repeat

euporie.core.key_binding.utils.**if_no_repeat**(*event: KeyPressEvent*) → bool

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

#### euporie.core.key_binding.utils.parse_keys

euporie.core.key_binding.utils.**parse_keys**(*keys: AnyKeys*) → list[tuple[str | *Keys*, ...]]

Pare a list of keys.

### Classes

| | |
|---|---|
| *Keys*(value[, names, module, qualname, type, ...]) | List of keys for use in key bindings. |

#### euporie.core.key_binding.utils.Keys

**class** euporie.core.key_binding.utils.**Keys**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

List of keys for use in key bindings.

Note that this is an "StrEnum", all values can be compared against strings.

euporie.core.key_binding.utils.**format_keys**(*keys: list[tuple[str | prompt_toolkit.keys.Keys, ...]]*) → list[str]

Convert a list of tuples of keys to a string.

euporie.core.key_binding.utils.**if_no_repeat**(*event: KeyPressEvent*) → bool

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

euporie.core.key_binding.utils.**parse_keys**(*keys: AnyKeys*) → list[tuple[str | *Keys*, ...]]

Pare a list of keys.

### euporie.core.key_binding.vi_state

Create a Vi state which defaults to navigation mode.

### Classes

| | |
|---|---|
| *InputMode*(value[, names, module, qualname, ...]) | |
| *PtkViState* | alias of ViState |
| *ViState*() | Mutable class to hold the state of the Vi navigation. |

### euporie.core.key_binding.vi_state.InputMode

**class** euporie.core.key_binding.vi_state.**InputMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.key_binding.vi_state.PtkViState

euporie.core.key_binding.vi_state.**PtkViState**

alias of ViState

### euporie.core.key_binding.vi_state.ViState

**class** euporie.core.key_binding.vi_state.**ViState**

Mutable class to hold the state of the Vi navigation.

**class** euporie.core.key_binding.vi_state.**ViState**

Bases: ViState

Mutable class to hold the state of the Vi navigation.

**property input_mode: *InputMode***

Get *InputMode*.

**last_character_find: CharacterFind | None**

None or CharacterFind instance. (This is used to repeat the last search in Vi mode, by pressing the 'n' or 'N' in navigation mode.)

**named_registers: dict[str, *ClipboardData*]**

Named registers. Maps register name (e.g. 'a') to ClipboardData instances.

**reset**() → None

Reset state, go back to the given mode. NAVIGATION by default.

**tilde_operator**

When true, make ~ act as an operator.

**waiting_for_digraph**

Waiting for digraph.

### euporie.core.keys

Register additional key escape sequences.

### Functions

| | |
|---|---|
| *extend_enum*(enumeration, name, *args, **kwds) | Add a new member to an existing Enum. |

### euporie.core.keys.extend_enum

euporie.core.keys.**extend_enum**(*enumeration*, *name*, **args*, ***kwds*)

Add a new member to an existing Enum.

### Classes

| | |
|---|---|
| *Keys*(value[, names, module, qualname, type, ...]) | List of keys for use in key bindings. |
| *combinations*(iterable, r) | Return successive r-length combinations of elements in the iterable. |

### euporie.core.keys.Keys

**class** euporie.core.keys.**Keys**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

List of keys for use in key bindings.

Note that this is an "StrEnum", all values can be compared against strings.

### euporie.core.keys.combinations

**class** euporie.core.keys.**combinations**(*iterable*, *r*)

Return successive r-length combinations of elements in the iterable.

combinations(range(4), 3) –> (0,1,2), (0,1,3), (0,2,3), (1,2,3)

### euporie.core.launch

Define a simple app for launching euporie apps.

## Functions

| | |
|---|---|
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *entry_points*(**params) | Return EntryPoint objects for all installed packages. |
| *setup_logs*([config]) | Configure the logger for euporie. |

### euporie.core.launch.add_setting

euporie.core.launch.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.launch.entry_points

euporie.core.launch.**entry_points**(***params*) → EntryPoints

Return EntryPoint objects for all installed packages.

Pass selection parameters (group or name) to filter the result to entry points matching those properties (see EntryPoints.select()).

**Returns**
EntryPoints for all installed packages.

### euporie.core.launch.setup_logs

euporie.core.launch.**setup_logs**(*config: Config | None = None*) → None

Configure the logger for euporie.

## Classes

| | |
|---|---|
| *Config*() | A configuration store. |
| *CoreApp*() | Launch a euporie application. |

### euporie.core.launch.Config

**class** euporie.core.launch.**Config**

A configuration store.

### euporie.core.launch.CoreApp

**class** euporie.core.launch.**CoreApp**

Launch a euporie application.

**class** euporie.core.launch.**CoreApp**

Bases: object

Launch a euporie application.

**classmethod launch**() → None

Launch the app.

**log_stdout_level = 'error'**

**name = 'launch'**

### euporie.core.layout

Contains containers and controls relating to layout.

### Modules

| | |
|---|---|
| *euporie.core.layout.cache* | Defines a container which saves rendered output and re-displays it. |
| *euporie.core.layout.containers* | Overrides for PTK containers which only render visible lines. |
| *euporie.core.layout.controls* | Miscellaneous control fields. |
| *euporie.core.layout.decor* | Decorative widgets. |
| *euporie.core.layout.mouse* | Defines a container which displays all children at full height vertially stacked. |
| *euporie.core.layout.print* | Defines a container which displays all children at full height vertially stacked. |
| *euporie.core.layout.screen* | Overrides for PTK containers which only render visible lines. |
| *euporie.core.layout.scroll* | Contains containers which display children at full height vertially stacked. |

### euporie.core.layout.cache

Defines a container which saves rendered output and re-displays it.

### Functions

| | |
|---|---|
| *get_app*() | Get the current active (running) Application. |
| *lru_cache*([maxsize, typed]) | Least-recently-used cache decorator. |
| *to_container*(container) | Make sure that the given object is a *Container*. |
| *walk*(container[, skip_hidden]) | Walk through layout, starting at this container. |

#### euporie.core.layout.cache.get_app

euporie.core.layout.cache.**get_app**() → *BaseApp*

Get the current active (running) Application.

#### euporie.core.layout.cache.lru_cache

euporie.core.layout.cache.**lru_cache**(*maxsize=128*, *typed=False*)

Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

#### euporie.core.layout.cache.to_container

euporie.core.layout.cache.**to_container**(*container: AnyContainer*) → *Container*

Make sure that the given object is a *Container*.

#### euporie.core.layout.cache.walk

euporie.core.layout.cache.**walk**(*container:* Container, *skip_hidden: bool = False*) → Iterable[*Container*]

Walk through layout, starting at this container.

### Classes

| | |
|---|---|
| *BoundedWritePosition*(xpos, ypos, width, height) | A write position which also hold bounding box information. |
| *CachedContainer*(content[, mouse_handler_wrapper]) | A container which renders its content once and caches the output. |
| *Container*() | Base class for user interface layout. |
| *DiInt*([top, right, bottom, left]) | A tuple of four integers with directions. |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, sent to *UIControl.mouse_handler*. |
| *MouseHandlers*() | Two dimensional raster of callbacks for mouse events. |
| *Point*(x, y) | |
| *Screen*([default_char, initial_width, ...]) | Screen class which uses :py:`BoundedWritePosition`s. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *WindowRenderInfo*(window, ui_content, ...) | Render information for the last render time of this control. |

#### euporie.core.layout.cache.BoundedWritePosition

**class** euporie.core.layout.cache.**BoundedWritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*, *bbox:* euporie.core.data_structures.DiInt | *None* = *None*)

A write position which also hold bounding box information.

#### euporie.core.layout.cache.CachedContainer

**class** euporie.core.layout.cache.**CachedContainer**(*content: AnyContainer*, *mouse_handler_wrapper: Callable[[MouseHandler,* CachedContainer*], MouseHandler]* | *None* = *None*)

A container which renders its content once and caches the output.

#### euporie.core.layout.cache.Container

**class** euporie.core.layout.cache.**Container**

Base class for user interface layout.

#### euporie.core.layout.cache.DiInt

**class** euporie.core.layout.cache.**DiInt**(*top: int* = *0*, *right: int* = *0*, *bottom: int* = *0*, *left: int* = *0*)

A tuple of four integers with directions.

### euporie.core.layout.cache.MouseEvent

**class** euporie.core.layout.cache.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers:* *frozenset*[prompt_toolkit.mouse_events.MouseModifier])

Mouse event, sent to *UIControl.mouse_handler*.

**Parameters**

- **position** – *Point* instance.
- **event_type** – *MouseEventType*.

### euporie.core.layout.cache.MouseHandlers

**class** euporie.core.layout.cache.**MouseHandlers**

Two dimensional raster of callbacks for mouse events.

### euporie.core.layout.cache.Point

**class** euporie.core.layout.cache.**Point**(*x*, *y*)

### euporie.core.layout.cache.Screen

**class** euporie.core.layout.cache.**Screen**(*default_char:* prompt_toolkit.layout.screen.Char | *None* = *None*, *initial_width: int = 0*, *initial_height: int = 0*)

Screen class which uses :py:`BoundedWritePosition`s.

### euporie.core.layout.cache.Window

**class** euporie.core.layout.cache.**Window**(*content:* UIControl | *None* = *None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int* | *None* = *None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]] = None*, *align:* WindowAlign | *Callable[[],* WindowAlign*] = WindowAlign.LEFT*, *style: str* | *Callable[[], str] = ''*, *char: None* | *str* | *Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable* | *None = None*)

Container that holds a control.

**Parameters**

- **content** – *UIControl* instance.
- **width** – *Dimension* instance or callable.
- **height** – *Dimension* instance or callable.
- **z_index** – When specified, this can be used to bring element in front of floating elements.
- **dont_extend_width** – When *True*, don't take up more width then the preferred width reported by the control.
- **dont_extend_height** – When *True*, don't take up more width then the preferred height reported by the control.
- **ignore_content_width** – A *bool* or *Filter* instance. Ignore the *UIContent* width when calculating the dimensions.
- **ignore_content_height** – A *bool* or *Filter* instance. Ignore the *UIContent* height when calculating the dimensions.
- **left_margins** – A list of *Margin* instance to be displayed on the left. For instance: NumberedMargin can be one of them in order to show line numbers.
- **right_margins** – Like *left_margins*, but on the other side.
- **scroll_offsets** – *ScrollOffsets* instance, representing the preferred amount of lines/columns to be always visible before/after the cursor. When both top and bottom are a very high number, the cursor will be centered vertically most of the time.
- **allow_scroll_beyond_bottom** – A *bool* or *Filter* instance. When True, allow scrolling so far, that the top part of the content is not visible anymore, while there is still empty space available at the bottom of the window. In the Vi editor for instance, this is possible. You will see tildes while the top part of the body is hidden.
- **wrap_lines** – A *bool* or *Filter* instance. When True, don't scroll horizontally, but wrap lines instead.
- **get_vertical_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll. (When this is *None*, the scroll is only determined by the last and current cursor position.)
- **get_horizontal_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll.
- **always_hide_cursor** – A *bool* or *Filter* instance. When True, never display the cursor, even when the user control specifies a cursor position.
- **cursorline** – A *bool* or *Filter* instance. When True, display a cursorline.
- **cursorcolumn** – A *bool* or *Filter* instance. When True, display a cursorcolumn.
- **colorcolumns** – A list of ColorColumn instances that describe the columns to be highlighted, or a callable that returns such a list.
- **align** – *WindowAlign* value or callable that returns an *WindowAlign* value. alignment of content.
- **style** – A style string. Style to be applied to all the cells in this window. (This can be a callable that returns a string.)

- **char** – (string) Character to be used for filling the background. This can also be a callable that returns a character.
- **get_line_prefix** – None or a callable that returns formatted text to be inserted before a line. It takes a line number (int) and a wrap_count and returns formatted text. This can be used for implementation of line continuations, things like Vim "breakindent" and so on.

### euporie.core.layout.cache.WindowRenderInfo

**class** euporie.core.layout.cache.**WindowRenderInfo**(*window:* Window, *ui_content:* UIContent, *horizontal_scroll:* *int*, *vertical_scroll:* *int*, *window_width:* *int*, *window_height:* *int*, *configured_scroll_offsets:* ScrollOffsets, *visible_line_to_row_col:* *dict[int, tuple[int, int]]*, *rowcol_to_yx:* *dict[tuple[int, int], tuple[int, int]]*, *x_offset:* *int*, *y_offset:* *int*, *wrap_lines:* *bool*)

Render information for the last render time of this control. It stores mapping information between the input buffers (in case of a *BufferControl*) and the actual render position on the output screen.

(Could be used for implementation of the Vi 'H' and 'L' key bindings as well as implementing mouse support.)

**Parameters**

- **ui_content** – The original *UIContent* instance that contains the whole input, without clipping. (ui_content)
- **horizontal_scroll** – The horizontal scroll of the *Window* instance.
- **vertical_scroll** – The vertical scroll of the *Window* instance.
- **window_width** – The width of the window that displays the content, without the margins.
- **window_height** – The height of the window that displays the content.
- **configured_scroll_offsets** – The scroll offsets as configured for the *Window* instance.
- **visible_line_to_row_col** – Mapping that maps the row numbers on the displayed screen (starting from zero for the first visible line) to (row, col) tuples pointing to the row and column of the *UIContent*.
- **rowcol_to_yx** – Mapping that maps (row, column) tuples representing coordinates of the UIContent to (y, x) absolute coordinates at the rendered screen.

**class** euporie.core.layout.cache.**CachedContainer**(*content: AnyContainer*, *mouse_handler_wrapper: Callable[[MouseHandler,* CachedContainer*], MouseHandler]* | *None* = *None*)

Bases: *Container*

A container which renders its content once and caches the output.

**blit**(*screen: PtkScreen*, *mouse_handlers:* MouseHandlers, *left:* *int*, *top:* *int*, *cols:* *slice*, *rows:* *slice*) → None

Copy 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 handelers 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 columns to copy
- **rows** – The rows to copy .

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

Return a list of all child containers.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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.

**invalidate**() → None

Flag the child's rendering as out-of-date.

**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 layout_hash: int**

Return a hash of the child's current layout.

**preferred_height**(*width: int*, *max_available_height: int*) → *Dimension*

Return the desired height for this container.

**preferred_width**(*max_available_width: int*) → *Dimension*

Return the desired width for this container.

**render**(*available_width: int*, *available_height: int*, *style: str = ''*, *start: int | None = None*, *end: int | None = None*) → None

Render 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
- **bbox** – The bounding box for the content to render
- **start** – Rows between top of output and top of scrollable pane
- **end** – Rows between top of output and bottom of scrollable pane

**reset**() → None

Reset the state of this container.

**write_to_screen**(*screen: PtkScreen*, *mouse_handlers:* MouseHandlers, *write_position:* WritePosition, *parent_style: str*, *erase_bg: bool*, *z_index: int | None*) → None

Render the container to a *Screen* instance.

**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.

### euporie.core.layout.containers

Overrides for PTK containers which only render visible lines.

### Functions

| | |
|---|---|
| *explode_text_fragments*(fragments) | Turn a list of (style_str, text) tuples into another list where each string is exactly one character. |
| *fragment_list_width*(fragments) | Return the character width of this text fragment list. |
| *get_app*() | Get the current active (running) Application. |
| *get_cwidth*(string) | Return width of a string. |
| *lru_cache*([maxsize, typed]) | Least-recently-used cache decorator. |
| *sum_layout_dimensions*(dimensions) | Sum a list of *Dimension* instances. |
| *take_using_weights*(items, weights) | Generator that keeps yielding items from the items list, in proportion to their weight. For instance::. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |
| *to_str*(value) | Turn callable or string into string. |

### euporie.core.layout.containers.explode_text_fragments

euporie.core.layout.containers.**explode_text_fragments**(*fragments: Iterable[_T]*) → _ExplodedList[_T]

Turn a list of (style_str, text) tuples into another list where each string is exactly one character.

It should be fine to call this function several times. Calling this on a list that is already exploded, is a null operation.

**Parameters**

**fragments** – List of (style, text) tuples.

### euporie.core.layout.containers.fragment_list_width

euporie.core.layout.containers.**fragment_list_width**(*fragments: StyleAndTextTuples*) → int

Return the character width of this text fragment list. (Take double width characters into account.)

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.layout.containers.get_app

euporie.core.layout.containers.**get_app**() → *Application*[Any]

Get the current active (running) Application. An *Application* is active during the Application.run_async() call.

We assume that there can only be one *Application* active at the same time. There is only one terminal window, with only one stdin and stdout. This makes the code significantly easier than passing around the *Application* everywhere.

If no *Application* is running, then return by default a DummyApplication. For practical reasons, we prefer to not raise an exception. This way, we don't have to check all over the place whether an actual *Application* was returned.

(For applications like pymux where we can have more than one *Application*, we'll use a work-around to handle that.)

### euporie.core.layout.containers.get_cwidth

euporie.core.layout.containers.**get_cwidth**(*string: str*) → int

Return width of a string. Wrapper around wcwidth.

### euporie.core.layout.containers.lru_cache

euporie.core.layout.containers.**lru_cache**(*maxsize=128*, *typed=False*)

Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

### euporie.core.layout.containers.sum_layout_dimensions

euporie.core.layout.containers.**sum_layout_dimensions**(*dimensions: list[*prompt_toolkit.layout.dimension.Dimension*]*) → *Dimension*

Sum a list of *Dimension* instances.

### euporie.core.layout.containers.take_using_weights

euporie.core.layout.containers.**take_using_weights**(*items: list[_T]*, *weights: list[int]*) → Generator[_T, None, None]

Generator that keeps yielding items from the items list, in proportion to their weight. For instance:

```
# Getting the first 70 items from this generator should have yielded 10
# times A, 20 times B and 40 times C, all distributed equally..
take_using_weights(['A', 'B', 'C'], [5, 10, 20])
```

**Parameters**

- **items** – List of items to take from.
- **weights** – Integers representing the weight. (Numbers have to be integers, not floats.)

### euporie.core.layout.containers.to_formatted_text

euporie.core.layout.containers.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

### euporie.core.layout.containers.to_str

euporie.core.layout.containers.**to_str**(*value: Union[Callable[[], str], str]*) → str

Turn callable or string into string.

### Classes

| | |
|---|---|
| *BoundedWritePosition*(xpos, ypos, width, height) | A write position which also hold bounding box information. |
| *DiInt*([top, right, bottom, left]) | A tuple of four integers with directions. |
| *FloatContainer*(content, floats[, modal, ...]) | A *FloatContainer* which uses :py`BoundedWritePosition`s. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, sent to *UIControl.mouse_handler*. |
| *Point*(x, y) | |
| *UIContent*(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *WindowAlign*(value[, names, module, ...]) | Alignment of the Window content. |
| *WindowRenderInfo*(window, ui_content, ...) | Render information for the last render time of this control. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

#### euporie.core.layout.containers.BoundedWritePosition

**class** euporie.core.layout.containers.**BoundedWritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*, *bbox:* euporie.core.data_structures.DiInt | *None* = *None*)

A write position which also hold bounding box information.

#### euporie.core.layout.containers.DiInt

**class** euporie.core.layout.containers.**DiInt**(*top: int = 0*, *right: int = 0*, *bottom: int = 0*, *left: int = 0*)

A tuple of four integers with directions.

#### euporie.core.layout.containers.FloatContainer

**class** euporie.core.layout.containers.**FloatContainer**(*content: AnyContainer*, *floats: list[*Float*]*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str | Callable[[], str] = ''*, *z_index: int | None = None*)

A *FloatContainer* which uses :py`BoundedWritePosition`s.

### euporie.core.layout.containers.FormattedTextControl

**class** euporie.core.layout.containers.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **key_bindings** – a *KeyBindings* object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.layout.containers.HSplit

**class** euporie.core.layout.containers.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

### euporie.core.layout.containers.MouseEvent

**class** euporie.core.layout.containers.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers:* *frozenset*[prompt_toolkit.mouse_events.MouseModifier])

Mouse event, sent to *UIControl.mouse_handler*.

**Parameters**

- **position** – *Point* instance.
- **event_type** – *MouseEventType*.

### euporie.core.layout.containers.Point

**class** euporie.core.layout.containers.**Point**(*x*, *y*)

### euporie.core.layout.containers.UIContent

**class** euporie.core.layout.containers.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>, line_count: int = 0, cursor_position: Point | None = None, menu_position: Point | None = None, show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.layout.containers.VSplit

**class** euporie.core.layout.containers.**VSplit**(*children: Sequence[AnyContainer], window_too_small:* Container | *None = None, align: HorizontalAlign = HorizontalAlign.JUSTIFY, padding: AnyDimension = 0, padding_char: str* | *None = None, padding_style: str = '', width: AnyDimension = None, height: AnyDimension = None, z_index: int* | *None = None, modal: bool = False, key_bindings:* KeyBindingsBase | *None = None, style: str* | *Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.layout.containers.Window

**class** euporie.core.layout.containers.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.core.layout.containers.WindowAlign

**class** euporie.core.layout.containers.**WindowAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of the Window content.

Note that this is different from *HorizontalAlign* and *VerticalAlign*, which are used for the alignment of the child containers in respectively *VSplit* and *HSplit*.

### euporie.core.layout.containers.WindowRenderInfo

**class** euporie.core.layout.containers.**WindowRenderInfo**(*window: Window*, *ui_content: UIContent*, *horizontal_scroll: int*, *vertical_scroll: int*, *window_width: int*, *window_height: int*, *configured_scroll_offsets: ScrollOffsets*, *visible_line_to_row_col: dict[int, tuple[int, int]]*, *rowcol_to_yx: dict[tuple[int, int], tuple[int, int]]*, *x_offset: int*, *y_offset: int*, *wrap_lines: bool*)

Render information for the last render time of this control. It stores mapping information between the input buffers (in case of a *BufferControl*) and the actual render position on the output screen.

(Could be used for implementation of the Vi 'H' and 'L' key bindings as well as implementing mouse support.)

**Parameters**

- **ui_content** – The original *UIContent* instance that contains the whole input, without clipping. (ui_content)
- **horizontal_scroll** – The horizontal scroll of the *Window* instance.
- **vertical_scroll** – The vertical scroll of the *Window* instance.
- **window_width** – The width of the window that displays the content, without the margins.
- **window_height** – The height of the window that displays the content.
- **configured_scroll_offsets** – The scroll offsets as configured for the *Window* instance.
- **visible_line_to_row_col** – Mapping that maps the row numbers on the displayed screen (starting from zero for the first visible line) to (row, col) tuples pointing to the row and column of the *UIContent*.
- **rowcol_to_yx** – Mapping that maps (row, column) tuples representing coordinates of the *UIContent* to (y, x) absolute coordinates at the rendered screen.

### euporie.core.layout.containers.partial

**class** euporie.core.layout.containers.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.layout.containers.**FloatContainer**(*content: AnyContainer*, *floats: list[Float]*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*, *z_index: int | None = None*)

Bases: *FloatContainer*

A *FloatContainer* which uses :py`BoundedWritePosition`s.

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

Return the list of child *Container* objects.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Return the preferred height of the float container. (We don't care about the height of the floats, they should always fit into the dimensions provided by the container.)

**preferred_width**(*max_available_width: int*) → *Dimension*

Return a `Dimension` that represents the desired width for this container.

**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: int* | *None*) → None

Write the actual content to the screen.

**Parameters**

- **screen** – *Screen*
- **mouse_handlers** – *MouseHandlers*.
- **parent_style** – Style string to pass to the *Window* object. This will be applied to all content of the windows. *VSplit* and *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.core.layout.containers.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: VerticalAlign* = *VerticalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Bases: *HSplit*

Several layouts, one stacked above/under the other.

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

Return the list of child *Container* objects.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Return a `Dimension` that represents the desired height for this container.

**preferred_width**(*max_available_width: int*) → *Dimension*

Return a `Dimension` that represents the desired width for this container.

**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: int | None*) → None

Render the prompt to a *Screen* instance.

**Parameters**

**screen** – The *Screen* class to which the output has to be written.

**class** euporie.core.layout.containers.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str | Callable[[], str] = ''*)

Bases: *VSplit*

Several layouts, one stacked left/right of the other.

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

Return the list of child *Container* objects.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Return a Dimension that represents the desired height for this container.

**preferred_width**(*max_available_width: int*) → *Dimension*

Return a Dimension that represents the desired width for this container.

**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: int | None*) → None

Render the prompt to a *Screen* instance.

**Parameters**

**screen** – The *Screen* class to which the output has to be written.

**class** euporie.core.layout.containers.**Window**(*content:* UIControl | *None* = *None*, *width:* *AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *dont_extend_width: FilterOrBool* = *False*, *dont_extend_height: FilterOrBool* = *False*, *ignore_content_width: FilterOrBool* = *False*, *ignore_content_height: FilterOrBool* = *False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool* = *False*, *wrap_lines: FilterOrBool* = *False*, *get_vertical_scroll: Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool* = *False*, *cursorline: FilterOrBool* = *False*, *cursorcolumn: FilterOrBool* = *False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]]* = *None*, *align:* WindowAlign | *Callable[[],* WindowAlign*]* = *WindowAlign.LEFT*, *style: str* | *Callable[[], str]* = '', *char: None* | *str* | *Callable[[], str]* = None, *get_line_prefix: GetLinePrefixCallable* | *None* = *None*)

Bases: *Window*

Container that holds a control.

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

Return the list of child *Container* objects.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Calculate the preferred height for this window.

**preferred_width**(*max_available_width: int*) → *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: int* | *None*) → None

Write window to screen.

### euporie.core.layout.controls

Miscellaneous control fields.

### Classes

| | |
|---|---|
| *DummyControl*() | A dummy control object that doesn't paint any content. |
| *FocusableDummyControl*() | A dummy control object that doesn't paint any content. |
| *UIContent*(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| *UIControl*() | Base class for all user interface controls. |

### euporie.core.layout.controls.DummyControl

**class** euporie.core.layout.controls.**DummyControl**

A dummy control object that doesn't paint any content.

### euporie.core.layout.controls.FocusableDummyControl

**class** euporie.core.layout.controls.**FocusableDummyControl**

A dummy control object that doesn't paint any content.

### euporie.core.layout.controls.UIContent

**class** euporie.core.layout.controls.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>, line_count: int = 0, cursor_position: Point | None = None, menu_position: Point | None = None, show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.layout.controls.UIControl

**class** euporie.core.layout.controls.**UIControl**

Base class for all user interface controls.

**class** euporie.core.layout.controls.**DummyControl**

Bases: *UIControl*

A dummy control object that doesn't paint any content.

**create_content**(*width: int*, *height: int*) → *UIContent*

Return one blank line only.

**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**() → *KeyBindingsBase* | None

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.

When *NotImplemented* is returned, it means that the given event is not handled by the *UIControl* itself. The *Window* or key bindings can decide to handle this event as scrolling or changing focus.

**Parameters**
**mouse_event** – *MouseEvent* instance.

**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:* *GetLinePrefixCallable* | *None*) → int | None

**preferred_width**(*max_available_width: int*) → int | None

**reset**() → None

**class** euporie.core.layout.controls.**FocusableDummyControl**

Bases: *DummyControl*

A dummy control object that doesn't paint any content.

**create_content**(*width: int*, *height: int*) → *UIContent*

Return one blank line only.

**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**() → *KeyBindingsBase* | None

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

Make this control focusable.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

When *NotImplemented* is returned, it means that the given event is not handled by the *UIControl* itself. The *Window* or key bindings can decide to handle this event as scrolling or changing focus.

**Parameters**

**mouse_event** – *MouseEvent* instance.

**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: GetLinePrefixCallable* | *None*) → int | None

**preferred_width**(*max_available_width: int*) → int | None

**reset**() → None

### euporie.core.layout.decor

Decorative widgets.

### Functions

| | |
|---|---|
| *get_app*() | Get the current active (running) Application. |
| *has_focus*(value) | Enable when this buffer has the focus. |
| *to_container*(container) | Make sure that the given object is a *Container*. |

### euporie.core.layout.decor.get_app

euporie.core.layout.decor.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.layout.decor.has_focus

euporie.core.layout.decor.**has_focus**(*value: FocusableElement*) → *Condition*

Enable when this buffer has the focus.

### euporie.core.layout.decor.to_container

euporie.core.layout.decor.**to_container**(*container: AnyContainer*) → *Container*

Make sure that the given object is a *Container*.

### Classes

| | |
|---|---|
| *Char*([char, style]) | Represent a single character in a *Screen*. |
| *ColorPaletteColor*(base[, _base_override]) | A representation of a color with adjustment methods. |
| *Container*() | Base class for user interface layout. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *DropShadow*([amount]) | A transparent container which makes the background darker. |
| *FastDictCache*(get_value[, size]) | Fast, lightweight cache which keeps at most *size* items. |
| *FocusedStyle*(body[, style_focus, style_hover]) | Apply a style to child containers when focused or hovered. |
| *Line*([char, width, height, collapse, style]) | Draw a horizontal or vertical line. |
| *MouseEventType*(value[, names, module, ...]) | |
| *Pattern*(char[, pattern]) | Fill an area with a repeating background pattern. |
| *Screen*([default_char, initial_width, ...]) | Two dimensional buffer of *Char* instances. |
| *WritePosition*(xpos, ypos, width, height) | |

### euporie.core.layout.decor.Char

**class** euporie.core.layout.decor.**Char**(*char: str = ' '*, *style: str = ''*)

Represent a single character in a *Screen*.

This should be considered immutable.

**Parameters**

- **char** – A single character (can be a double-width character).
- **style** – A style string. (Can contain classnames.)

### euporie.core.layout.decor.ColorPaletteColor

**class** euporie.core.layout.decor.**ColorPaletteColor**(*base: str*, *_base_override: str = ''*)

A representation of a color with adjustment methods.

### euporie.core.layout.decor.Container

**class** euporie.core.layout.decor.**Container**

Base class for user interface layout.

### euporie.core.layout.decor.Dimension

**class** euporie.core.layout.decor.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.layout.decor.DropShadow

**class** euporie.core.layout.decor.**DropShadow**(*amount: float = 0.5*)

A transparent container which makes the background darker.

### euporie.core.layout.decor.FastDictCache

**class** euporie.core.layout.decor.**FastDictCache**(*get_value: Callable[[...], _V]*, *size: int = 1000000*)

Fast, lightweight cache which keeps at most *size* items. It will discard the oldest items in the cache first.

The cache is a dictionary, which doesn't keep track of access counts. It is perfect to cache little immutable objects which are not expensive to create, but where a dictionary lookup is still much faster than an object instantiation.

**Parameters**

**get_value** – Callable that's called in case of a missing key.

### euporie.core.layout.decor.FocusedStyle

**class** euporie.core.layout.decor.**FocusedStyle**(*body: AnyContainer*, *style_focus: str | Callable[[], str] = 'class:focused'*, *style_hover: str | Callable[[], str] = ''*)

Apply a style to child containers when focused or hovered.

### euporie.core.layout.decor.Line

**class** euporie.core.layout.decor.**Line**(*char: str | None = None*, *width: int | None = None*, *height: int | None = None*, *collapse: bool = False*, *style: str = 'class:grid-line'*)

Draw a horizontal or vertical line.

### euporie.core.layout.decor.MouseEventType

**class** euporie.core.layout.decor.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.layout.decor.Pattern

**class** euporie.core.layout.decor.**Pattern**(*char: str | Callable[[], str]*, *pattern: int | Callable[[], int] = 1*)

Fill an area with a repeating background pattern.

### euporie.core.layout.decor.Screen

**class** euporie.core.layout.decor.**Screen**(*default_char: prompt_toolkit.layout.screen.Char | None = None*, *initial_width: int = 0*, *initial_height: int = 0*)

Two dimensional buffer of *Char* instances.

### euporie.core.layout.decor.WritePosition

**class** euporie.core.layout.decor.**WritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*)

**class** euporie.core.layout.decor.**DropShadow**(*amount: float = 0.5*)

Bases: *Container*

A transparent container which makes the background darker.

**property cp: *ColorPalette***

Get the current app's current color palette.

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

Return an empty list of child *Container* objects.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Return the wrapped container's preferred height.

**preferred_width**(*max_available_width: int*) → *Dimension*

Return the wrapped container's preferred width.

**reset**() → None

Reset the wrapped container - here, do nothing.

**write_to_screen**(*screen:* Screen, *mouse_handlers:* MouseHandlers, *write_position:* WritePosition, *parent_style: str*, *erase_bg: bool*, *z_index: int* | *None*) → None

Draw the wrapped container with the additional style.

**class** euporie.core.layout.decor.**FocusedStyle**(*body: AnyContainer*, *style_focus: str* | *Callable[[], str] = 'class:focused'*, *style_hover: str* | *Callable[[], str] = ''*)

Bases: `Container`

Apply a style to child containers when focused or hovered.

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

Return the list of child `Container` objects.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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.

**get_style**() → str

Determine the style to apply depending on the focus status.

**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*) → *Dimension*

Return the wrapped container's preferred height.

**preferred_width**(*max_available_width: int*) → *Dimension*

Return the wrapped container's preferred width.

**reset**() → None

Reset the wrapped container.

**write_to_screen**(*screen:* Screen, *mouse_handlers:* MouseHandlers, *write_position:* WritePosition, *parent_style: str*, *erase_bg: bool*, *z_index: int* | *None*) → None

Draw the wrapped container with the additional style.

**class** euporie.core.layout.decor.**Line**(*char: str* | *None* = *None*, *width: int* | *None* = *None*, *height: int* | *None* = *None*, *collapse: bool* = *False*, *style: str* = *'class:grid-line'*)

Bases: *Container*

Draw a horizontal or vertical line.

**get_children**() → list

Return an empty list of the container's children.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Return the preferred height of the line.

**preferred_width**(*max_available_width: int*) → *Dimension*

Return the preferred width of the line.

**reset**() → None

Reset the state of the line. Does nothing.

**write_to_screen**(*screen:* Screen, *mouse_handlers:* MouseHandlers, *write_position:* WritePosition, *parent_style: str*, *erase_bg: bool*, *z_index: int* | *None*) → None

Draw 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.core.layout.decor.**Pattern**(*char: str* | *Callable[[], str]*, *pattern: int* | *Callable[[], int]* = *1*)

Bases: *Container*

Fill an area with a repeating background pattern.

**get_children**() → list

Return an empty list of the container's children.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Return an empty dimension (expand to available height).

**preferred_width**(*max_available_width: int*) → *Dimension*

Return an empty dimension (expand to available width).

**reset**() → None

Reset the pattern. Does nothing.

**write_to_screen**(*screen:* Screen, *mouse_handlers:* MouseHandlers, *write_position:* WritePosition, *parent_style: str*, *erase_bg: bool*, *z_index: int | None*) → 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.

## euporie.core.layout.mouse

Defines a container which displays all children at full height vertially stacked.

## Functions

| | |
|---|---|
| *get_app*() | Get the current active (running) Application. |
| *lru_cache*([maxsize, typed]) | Least-recently-used cache decorator. |
| *to_container*(container) | Monkey-patch *to_container* to collect container status functions. |

### euporie.core.layout.mouse.get_app

euporie.core.layout.mouse.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.layout.mouse.lru_cache

euporie.core.layout.mouse.**lru_cache**(*maxsize=128*, *typed=False*)

Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

### euporie.core.layout.mouse.to_container

euporie.core.layout.mouse.**to_container**(*container: AnyContainer*) → *Container*

Monkey-patch *to_container* to collect container status functions.

### Classes

| | |
|---|---|
| *Condition*(func) | Turn any callable into a Filter. |
| *Container*() | Base class for user interface layout. |
| *DisableMouseOnScroll*(content) | A container which disables mouse support on unhandled scroll up events. |
| *MouseEventType*(value[, names, module, ...]) | |

### euporie.core.layout.mouse.Condition

**class** euporie.core.layout.mouse.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.layout.mouse.Container

**class** euporie.core.layout.mouse.**Container**

Base class for user interface layout.

### euporie.core.layout.mouse.DisableMouseOnScroll

**class** euporie.core.layout.mouse.**DisableMouseOnScroll**(*content: AnyContainer*)

A container which disables mouse support on unhandled scroll up events.

This enables the terminal scroll-back buffer to be scrolled if there is nothing in the application which is scrollable.

### euporie.core.layout.mouse.MouseEventType

**class** euporie.core.layout.mouse.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

**class** euporie.core.layout.mouse.**DisableMouseOnScroll**(*content: AnyContainer*)

Bases: *Container*

A container which disables mouse support on unhandled scroll up events.

This enables the terminal scroll-back buffer to be scrolled if there is nothing in the application which is scrollable.

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

Return a list of all child containers.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Return the desired height for this container.

**preferred_width**(*max_available_width: int*) → *Dimension*

Return the desired width for this container.

**reset**() → None

Reset the state of this container.

**write_to_screen**(*screen:* Screen, *mouse_handlers:* MouseHandlers, *write_position:* WritePosition, *parent_style: str*, *erase_bg: bool*, *z_index: int* | *None*) → None

Render the container to a *Screen* instance.

Wrap mouse handelrs, hooking unhandled scroll up events so the terminal can be scrolled if the scroll event is not handled.

**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.

### euporie.core.layout.print

Defines a container which displays all children at full height vertially stacked.

### Functions

| | |
|---|---|
| *to_container*(container) | Make sure that the given object is a *Container*. |
| *to_dimension*(value) | Turn the given object into a *Dimension* object. |

### euporie.core.layout.print.to_container

euporie.core.layout.print.**to_container**(*container: AnyContainer*) → *Container*

Make sure that the given object is a *Container*.

### euporie.core.layout.print.to_dimension

euporie.core.layout.print.**to_dimension**(*value: Union[None, int,* Dimension*, Callable[[], Any]]*) → *Dimension*

Turn the given object into a *Dimension* object.

### Classes

| | |
|---|---|
| *BoundedWritePosition*(xpos, ypos, width, height) | A write position which also hold bounding box information. |
| *Container*() | Base class for user interface layout. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *PrintingContainer*(children[, width, ...]) | A container which displays all it's children in a vertical list. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |

### euporie.core.layout.print.BoundedWritePosition

**class** euporie.core.layout.print.**BoundedWritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*, *bbox:* euporie.core.data_structures.DiInt | *None* = *None*)

A write position which also hold bounding box information.

### euporie.core.layout.print.Container

**class** euporie.core.layout.print.**Container**

Base class for user interface layout.

### euporie.core.layout.print.Dimension

**class** euporie.core.layout.print.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.layout.print.PrintingContainer

**class** euporie.core.layout.print.**PrintingContainer**(*children: Callable | Sequence[AnyContainer]*, *width: AnyDimension = None*, *key_bindings:* KeyBindingsBase | *None* = *None*)

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

### euporie.core.layout.print.Window

**class** euporie.core.layout.print.**Window**(*content:* UIControl | *None* = *None*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *dont_extend_width: FilterOrBool* = *False*, *dont_extend_height: FilterOrBool* = *False*, *ignore_content_width: FilterOrBool* = *False*, *ignore_content_height: FilterOrBool* = *False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool* = *False*, *wrap_lines: FilterOrBool* = *False*, *get_vertical_scroll: Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool* = *False*, *cursorline: FilterOrBool* = *False*, *cursorcolumn: FilterOrBool* = *False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]]* = *None*, *align:* WindowAlign | *Callable[[],* WindowAlign*]* = *WindowAlign.LEFT*, *style: str* | *Callable[[], str]* = '', *char: None* | *str* | *Callable[[], str]* = *None*, *get_line_prefix: GetLinePrefixCallable* | *None* = *None*)

Container that holds a control.

**Parameters**

- **content** – *UIControl* instance.
- **width** – *Dimension* instance or callable.
- **height** – *Dimension* instance or callable.
- **z_index** – When specified, this can be used to bring element in front of floating elements.
- **dont_extend_width** – When *True*, don't take up more width then the preferred width reported by the control.
- **dont_extend_height** – When *True*, don't take up more width then the preferred height reported by the control.
- **ignore_content_width** – A *bool* or *Filter* instance. Ignore the *UIContent* width when calculating the dimensions.
- **ignore_content_height** – A *bool* or *Filter* instance. Ignore the *UIContent* height when calculating the dimensions.
- **left_margins** – A list of *Margin* instance to be displayed on the left. For instance: NumberedMargin can be one of them in order to show line numbers.
- **right_margins** – Like *left_margins*, but on the other side.
- **scroll_offsets** – *ScrollOffsets* instance, representing the preferred amount of lines/columns to be always visible before/after the cursor. When both top and bottom are a very high number, the cursor will be centered vertically most of the time.
- **allow_scroll_beyond_bottom** – A *bool* or *Filter* instance. When True, allow scrolling so far, that the top part of the content is not visible anymore, while there is still empty space available at the bottom of the window. In the Vi editor for instance, this is possible. You will see tildes while the top part of the body is hidden.

- **wrap_lines** – A *bool* or *Filter* instance. When True, don't scroll horizontally, but wrap lines instead.
- **get_vertical_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll. (When this is *None*, the scroll is only determined by the last and current cursor position.)
- **get_horizontal_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll.
- **always_hide_cursor** – A *bool* or *Filter* instance. When True, never display the cursor, even when the user control specifies a cursor position.
- **cursorline** – A *bool* or *Filter* instance. When True, display a cursorline.
- **cursorcolumn** – A *bool* or *Filter* instance. When True, display a cursorcolumn.
- **colorcolumns** – A list of ColorColumn instances that describe the columns to be highlighted, or a callable that returns such a list.
- **align** – *WindowAlign* value or callable that returns an *WindowAlign* value. alignment of content.
- **style** – A style string. Style to be applied to all the cells in this window. (This can be a callable that returns a string.)
- **char** – (string) Character to be used for filling the background. This can also be a callable that returns a character.
- **get_line_prefix** – None or a callable that returns formatted text to be inserted before a line. It takes a line number (int) and a wrap_count and returns formatted text. This can be used for implementation of line continuations, things like Vim "breakindent" and so on.

**class** euporie.core.layout.print.**PrintingContainer**(*children: Callable | Sequence[AnyContainer]*, *width: AnyDimension = None*, *key_bindings: KeyBindingsBase | None = None*)

Bases: *Container*

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

**property children: Sequence[AnyContainer]**

Return the container's children.

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

Return a list of all child containers.

**get_key_bindings**() → *KeyBindingsBase* | None

Return the container's key bindings.

**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*) → *Dimension*

Return the preferred height, equal to the sum of the child heights.

**preferred_width**(*max_available_width: int*) → *Dimension*

Calculate 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: int* | *None*) → 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.

## euporie.core.layout.screen

Overrides for PTK containers which only render visible lines.

### Classes

| | |
|---|---|
| *BoundedWritePosition*(xpos, ypos, width, height) | A write position which also hold bounding box information. |
| *DiInt*([top, right, bottom, left]) | A tuple of four integers with directions. |
| *Screen*([default_char, initial_width, ...]) | Screen class which uses :py:`BoundedWritePosition`s. |

## euporie.core.layout.screen.BoundedWritePosition

**class** euporie.core.layout.screen.**BoundedWritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*, *bbox:* euporie.core.data_structures.DiInt | *None* = *None*)

A write position which also hold bounding box information.

### euporie.core.layout.screen.DiInt

**class** euporie.core.layout.screen.**DiInt**(*top: int = 0*, *right: int = 0*, *bottom: int = 0*, *left: int = 0*)

A tuple of four integers with directions.

### euporie.core.layout.screen.Screen

**class** euporie.core.layout.screen.**Screen**(*default_char: prompt_toolkit.layout.screen.Char | None = None*, *initial_width: int = 0*, *initial_height: int = 0*)

Screen class which uses :py:`BoundedWritePosition`s.

**class** euporie.core.layout.screen.**BoundedWritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*, *bbox: euporie.core.data_structures.DiInt | None = None*)

Bases: *WritePosition*

A write position which also hold bounding box information.

**class** euporie.core.layout.screen.**Screen**(*default_char: prompt_toolkit.layout.screen.Char | None = None*, *initial_width: int = 0*, *initial_height: int = 0*)

Bases: *Screen*

Screen class which uses :py:`BoundedWritePosition`s.

**append_style_to_content**(*style_str: str*) → None

For all the characters in the screen. Set the style string to the given *style_str*.

**cursor_positions: dict[*Window*, *Point*]**

Position of the cursor.

**draw_all_floats**() → None

Draw all float functions in order of z-index.

**draw_with_z_index**(*z_index: int*, *draw_func: Callable[[], None]*) → None

Add a draw-function for a *Window* which has a >= 0 z_index. This will be postponed until *draw_all_floats* is called.

**fill_area**(*write_position: WritePosition*, *style: str = ''*, *after: bool = False*) → None

Fill the content of this area, using the given *style*.

**get_cursor_position**(*window: Window*) → *Point*

Get the cursor position for a given window. Returns a *Point*.

**get_menu_position**(*window: Window*) → *Point*

Get the menu position for a given window. (This falls back to the cursor position if no menu position was set.)

**menu_positions: dict[*Window*, *Point*]**

(Optional) Where to position the menu. E.g. at the start of a completion. (We can't use the cursor position, because we don't want the completion menu to change its position when we browse through all the completions.)

**set_cursor_position**(*window: Window*, *position: Point*) → None

Set the cursor position for a given window.

**set_menu_position**(*window:* Window, *position:* Point) → None

Set the cursor position for a given window.

**show_cursor**

Visibility of the cursor.

**property visible_windows: list[*Window*]**

**width**

Currently used width/height of the screen. This will increase when data is written to the screen.

**zero_width_escapes: *defaultdict*[int, *defaultdict*[int, str]]**

Escape sequences to be injected.

### euporie.core.layout.scroll

Contains containers which display children at full height vertially stacked.

### Functions

| | |
|---|---|
| *cast*(typ, val) | Cast a value to a type. |
| *get_app*() | Get the current active (running) Application. |
| *run_in_thread_with_context*(func, *args[, daemon]) | Run a function in an thread, but make sure it uses the same contextvars. |
| *to_container*(container) | Make sure that the given object is a *Container*. |
| *to_dimension*(value) | Turn the given object into a *Dimension* object. |

### euporie.core.layout.scroll.cast

euporie.core.layout.scroll.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.layout.scroll.get_app

euporie.core.layout.scroll.**get_app**() → *Application*[Any]

Get the current active (running) Application. An *Application* is active during the Application.run_async() call.

We assume that there can only be one *Application* active at the same time. There is only one terminal window, with only one stdin and stdout. This makes the code significantly easier than passing around the *Application* everywhere.

If no *Application* is running, then return by default a DummyApplication. For practical reasons, we prefer to not raise an exception. This way, we don't have to check all over the place whether an actual *Application* was returned.

(For applications like pymux where we can have more than one *Application*, we'll use a work-around to handle that.)

### euporie.core.layout.scroll.run_in_thread_with_context

euporie.core.layout.scroll.**run_in_thread_with_context**(*func: Callable*, **args: Any*, *daemon: bool = True*, ***kwargs: Any*) → None

Run a function in an thread, but make sure it uses the same contextvars.

This is required so that the function will see the right application.

### euporie.core.layout.scroll.to_container

euporie.core.layout.scroll.**to_container**(*container: AnyContainer*) → *Container*

Make sure that the given object is a *Container*.

### euporie.core.layout.scroll.to_dimension

euporie.core.layout.scroll.**to_dimension**(*value: Union[None, int,* Dimension*, Callable[[], Any]]*) → *Dimension*

Turn the given object into a *Dimension* object.

### Classes

| | |
|---|---|
| *BoundedWritePosition*(xpos, ypos, width, height) | A write position which also hold bounding box information. |
| *CachedContainer*(content[, mouse_handler_wrapper]) | A container which renders its content once and caches the output. |
| *Container*() | Base class for user interface layout. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, sent to *UIControl.mouse_handler*. |
| *MouseEventType*(value[, names, module, ...]) | |
| *MouseModifier*(value[, names, module, ...]) | |
| *PrintingContainer*(children[, width, ...]) | A container which displays all it's children in a vertical list. |
| *ScrollOffsets*([top, bottom, left, right]) | Scroll offsets for the *Window* class. |
| *ScrollingContainer*(children[, height, ...]) | A scrollable container which renders only the currently visible children. |
| *UIContent*(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *WindowRenderInfo*(window, ui_content, ...) | Render information for the last render time of this control. |

### euporie.core.layout.scroll.BoundedWritePosition

**class** euporie.core.layout.scroll.**BoundedWritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*, *bbox:* euporie.core.data_structures.DiInt | *None* = *None*)

A write position which also hold bounding box information.

### euporie.core.layout.scroll.CachedContainer

**class** euporie.core.layout.scroll.**CachedContainer**(*content: AnyContainer*, *mouse_handler_wrapper: Callable[[MouseHandler,* CachedContainer*], MouseHandler]* | *None* = *None*)

A container which renders its content once and caches the output.

### euporie.core.layout.scroll.Container

**class** euporie.core.layout.scroll.**Container**

Base class for user interface layout.

### euporie.core.layout.scroll.Dimension

**class** euporie.core.layout.scroll.**Dimension**(*min: int* | *None* = *None*, *max: int* | *None* = *None*, *weight: int* | *None* = *None*, *preferred: int* | *None* = *None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.layout.scroll.MouseEvent

**class** euporie.core.layout.scroll.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers:* *frozenset*[prompt_toolkit.mouse_events.MouseModifier])

Mouse event, sent to *UIControl.mouse_handler*.

**Parameters**

- **position** – *Point* instance.
- **event_type** – *MouseEventType*.

### euporie.core.layout.scroll.MouseEventType

**class** euporie.core.layout.scroll.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.layout.scroll.MouseModifier

**class** euporie.core.layout.scroll.**MouseModifier**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.layout.scroll.PrintingContainer

**class** euporie.core.layout.scroll.**PrintingContainer**(*children: Callable | Sequence[AnyContainer]*, *width: AnyDimension = None*, *key_bindings:* KeyBindingsBase | *None = None*)

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

### euporie.core.layout.scroll.ScrollOffsets

**class** euporie.core.layout.scroll.**ScrollOffsets**(*top: Union[int, Callable[[], int]] = 0*, *bottom: Union[int, Callable[[], int]] = 0*, *left: Union[int, Callable[[], int]] = 0*, *right: Union[int, Callable[[], int]] = 0*)

Scroll offsets for the `Window` class.

Note that left/right offsets only make sense if line wrapping is disabled.

### euporie.core.layout.scroll.ScrollingContainer

**class** euporie.core.layout.scroll.**ScrollingContainer**(*children: Callable[[], Sequence[AnyContainer]] | Sequence[AnyContainer], height: AnyDimension = None, width: AnyDimension = None, style: str | Callable[[], str] = '', scroll_offsets: ScrollOffsets | None = None*)

A scrollable container which renders only the currently visible children.

### euporie.core.layout.scroll.UIContent

**class** euporie.core.layout.scroll.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>, line_count: int = 0, cursor_position: Point | None = None, menu_position: Point | None = None, show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.layout.scroll.Window

**class** euporie.core.layout.scroll.**Window**(*content: UIControl | None = None, width: AnyDimension = None, height: AnyDimension = None, z_index: int | None = None, dont_extend_width: FilterOrBool = False, dont_extend_height: FilterOrBool = False, ignore_content_width: FilterOrBool = False, ignore_content_height: FilterOrBool = False, left_margins: Sequence[Margin] | None = None, right_margins: Sequence[Margin] | None = None, scroll_offsets: ScrollOffsets | None = None, allow_scroll_beyond_bottom: FilterOrBool = False, wrap_lines: FilterOrBool = False, get_vertical_scroll: Callable[[Window], int] | None = None, get_horizontal_scroll: Callable[[Window], int] | None = None, always_hide_cursor: FilterOrBool = False, cursorline: FilterOrBool = False, cursorcolumn: FilterOrBool = False, colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None, align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT, style: str | Callable[[], str] = '', char: None | str | Callable[[], str] = None, get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

**Parameters**

- **content** – *UIControl* instance.
- **width** – *Dimension* instance or callable.
- **height** – *Dimension* instance or callable.
- **z_index** – When specified, this can be used to bring element in front of floating elements.
- **dont_extend_width** – When *True*, don't take up more width then the preferred width reported by the control.
- **dont_extend_height** – When *True*, don't take up more width then the preferred height reported by the control.
- **ignore_content_width** – A *bool* or *Filter* instance. Ignore the *UIContent* width when calculating the dimensions.
- **ignore_content_height** – A *bool* or *Filter* instance. Ignore the *UIContent* height when calculating the dimensions.
- **left_margins** – A list of *Margin* instance to be displayed on the left. For instance: NumberedMargin can be one of them in order to show line numbers.
- **right_margins** – Like *left_margins*, but on the other side.
- **scroll_offsets** – *ScrollOffsets* instance, representing the preferred amount of lines/columns to be always visible before/after the cursor. When both top and bottom are a very high number, the cursor will be centered vertically most of the time.
- **allow_scroll_beyond_bottom** – A *bool* or *Filter* instance. When True, allow scrolling so far, that the top part of the content is not visible anymore, while there is still empty space available at the bottom of the window. In the Vi editor for instance, this is possible. You will see tildes while the top part of the body is hidden.
- **wrap_lines** – A *bool* or *Filter* instance. When True, don't scroll horizontally, but wrap lines instead.
- **get_vertical_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll. (When this is *None*, the scroll is only determined by the last and current cursor position.)
- **get_horizontal_scroll** – Callable that takes this window instance as input and returns a preferred vertical scroll.
- **always_hide_cursor** – A *bool* or *Filter* instance. When True, never display the cursor, even when the user control specifies a cursor position.
- **cursorline** – A *bool* or *Filter* instance. When True, display a cursorline.
- **cursorcolumn** – A *bool* or *Filter* instance. When True, display a cursorcolumn.
- **colorcolumns** – A list of ColorColumn instances that describe the columns to be highlighted, or a callable that returns such a list.
- **align** – *WindowAlign* value or callable that returns an *WindowAlign* value. alignment of content.
- **style** – A style string. Style to be applied to all the cells in this window. (This can be a callable that returns a string.)

- **char** – (string) Character to be used for filling the background. This can also be a callable that returns a character.
- **get_line_prefix** – None or a callable that returns formatted text to be inserted before a line. It takes a line number (int) and a wrap_count and returns formatted text. This can be used for implementation of line continuations, things like Vim "breakindent" and so on.

### euporie.core.layout.scroll.WindowRenderInfo

**class** euporie.core.layout.scroll.**WindowRenderInfo**(*window: Window*, *ui_content: UIContent*, *horizontal_scroll: int*, *vertical_scroll: int*, *window_width: int*, *window_height: int*, *configured_scroll_offsets: ScrollOffsets*, *visible_line_to_row_col: dict[int, tuple[int, int]]*, *rowcol_to_yx: dict[tuple[int, int], tuple[int, int]]*, *x_offset: int*, *y_offset: int*, *wrap_lines: bool*)

Render information for the last render time of this control. It stores mapping information between the input buffers (in case of a *BufferControl*) and the actual render position on the output screen.

(Could be used for implementation of the Vi 'H' and 'L' key bindings as well as implementing mouse support.)

**Parameters**

- **ui_content** – The original *UIContent* instance that contains the whole input, without clipping. (ui_content)
- **horizontal_scroll** – The horizontal scroll of the *Window* instance.
- **vertical_scroll** – The vertical scroll of the *Window* instance.
- **window_width** – The width of the window that displays the content, without the margins.
- **window_height** – The height of the window that displays the content.
- **configured_scroll_offsets** – The scroll offsets as configured for the *Window* instance.
- **visible_line_to_row_col** – Mapping that maps the row numbers on the displayed screen (starting from zero for the first visible line) to (row, col) tuples pointing to the row and column of the *UIContent*.
- **rowcol_to_yx** – Mapping that maps (row, column) tuples representing coordinates of the *UIContent* to (y, x) absolute coordinates at the rendered screen.

**class** euporie.core.layout.scroll.**PrintingContainer**(*children: Callable | Sequence[AnyContainer]*, *width: AnyDimension = None*, *key_bindings: KeyBindingsBase | None = None*)

Bases: *Container*

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

**property children: Sequence[AnyContainer]**

Return the container's children.

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

Return a list of all child containers.

**get_key_bindings**() → *KeyBindingsBase* | None

Return the container's key bindings.

**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*) → *Dimension*

Return the preferred height, equal to the sum of the child heights.

**preferred_width**(*max_available_width: int*) → *Dimension*

Calculate and returns the desired width for this container.

**reset**() → None

Reet the state of this container and all the children.

Does nothing as this container is used for dumping output.

**write_to_screen**(*screen: PtkScreen*, *mouse_handlers:* MouseHandlers, *write_position:* WritePosition, *parent_style: str*, *erase_bg: bool*, *z_index: int* | *None*) → 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.

**class** euporie.core.layout.scroll.**ScrollingContainer**(*children: Callable[[], Sequence[AnyContainer]]* | *Sequence[AnyContainer]*, *height: AnyDimension = None*, *width: AnyDimension = None*, *style: str* | *Callable[[], str] = ''*, *scroll_offsets:* ScrollOffsets | *None = None*)

Bases: *Container*

A scrollable container which renders only the currently visible children.

**all_children**() → Sequence[*Container*]

Return the list of all children of this container.

**get_child**(*index: int* | *None = None*) → *CachedContainer*

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[*prompt_toolkit.layout.containers.Container*]

Return the list of currently visible children to include in the layout.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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:* MouseEvent) → NotImplementedOrNone

Mouse handler to scroll the pane.

**pre_render_children**(*width: int*, *height: int*) → None

Render all unrendered children in a background thread.

**preferred_height**(*width: int*, *max_available_height: int*) → *Dimension*

Return the preferred height only if one is provided.

**preferred_width**(*max_available_width: int*) → *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*) → NotImplementedOrNone

Scroll up or down a number of rows.

**Parameters**
**n** – The number of rows to scroll, negative for up, positive for down

**Returns**

**NotImplemented is scrolling is not allowed, otherwise**
None

**scroll_to**(*index: int*, *anchor: Literal['top', 'bottom']* | *None* = *None*) → None

Scroll a child into view.

**Parameters**

- **index** – The child index to scroll into view
- **anchor** – Whether to scroll to the top or bottom the given child index

**select**(*index: int*, *extend: bool* = *False*, *position: int* | *None* = *None*, *scroll: bool* = *True*) → None

Select a child or adds it to the selection.

**Parameters**

- **index** – The index of the cell to select

- **extend** – If true, the selection will be extended to include the cell
- **position** – An optional cursor position index to apply to the cell input
- **scroll** – Whether to scroll the page

**property selected_indices: list[int]**

Return in indices of the currently selected children.

**property selected_slice: slice**

Return the currently selected slice.

**validate_slice**(*slice_: slice*) → slice

Ensure a slice describes a valid range of children.

**property vertical_scroll: int**

The best guess at the absolute vertical scroll position.

**write_to_screen**(*screen: PtkScreen*, *mouse_handlers:* MouseHandlers, *write_position:* WritePosition, *parent_style: str*, *erase_bg: bool*, *z_index: int | None*) → 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.

## euporie.core.lexers

Relating to lexers.

### Functions

| | |
|---|---|
| *detect_lexer*([text, path, language]) | Detect the pygments lexer for a file. |
| *get_lexer_by_name*(_alias, **options) | Return an instance of a *Lexer* subclass that has *alias* in its aliases list. |
| *get_lexer_for_filename*(_fn[, code]) | Get a lexer for a filename. |
| *guess_lexer*(_text, **options) | Return a *Lexer* subclass instance that's guessed from the text in *text*. |
| *guess_lexer_for_filename*(_fn, _text, **options) | As *guess_lexer()*, but only lexers which have a pattern in *filenames* or *alias_filenames* that matches *filename* are taken into consideration. |

### euporie.core.lexers.detect_lexer

euporie.core.lexers.**detect_lexer**(*text: str = ''*, *path:* Path | *None = None*, *language: str = ''*) → PygmentsLexerCls | None

Detect the pygments lexer for a file.

### euporie.core.lexers.get_lexer_by_name

euporie.core.lexers.**get_lexer_by_name**(*_alias*, ***options*)

Return an instance of a *Lexer* subclass that has *alias* in its aliases list. The lexer is given the *options* at its instantiation.

Will raise *pygments.util.ClassNotFound* if no lexer with that alias is found.

### euporie.core.lexers.get_lexer_for_filename

euporie.core.lexers.**get_lexer_for_filename**(*_fn*, *code=None*, ***options*)

Get a lexer for a filename.

Return a *Lexer* subclass instance that has a filename pattern matching *fn*. The lexer is given the *options* at its instantiation.

Raise *pygments.util.ClassNotFound* if no lexer for that filename is found.

If multiple lexers match the filename pattern, use their `analyse_text()` methods to figure out which one is more appropriate.

### euporie.core.lexers.guess_lexer

euporie.core.lexers.**guess_lexer**(*_text*, ***options*)

Return a *Lexer* subclass instance that's guessed from the text in *text*. For that, the *analyse_text()* method of every known lexer class is called with the text as argument, and the lexer which returned the highest value will be instantiated and returned.

*pygments.util.ClassNotFound* is raised if no lexer thinks it can handle the content.

### euporie.core.lexers.guess_lexer_for_filename

euporie.core.lexers.**guess_lexer_for_filename**(*_fn*, *_text*, ***options*)

As *guess_lexer()*, but only lexers which have a pattern in *filenames* or *alias_filenames* that matches *filename* are taken into consideration.

*pygments.util.ClassNotFound* is raised if no lexer thinks it can handle the content.

### Exceptions

| | |
|---|---|
| *ClassNotFound* | Raised if one of the lookup functions didn't find a matching class. |

### euporie.core.lexers.ClassNotFound

**exception** euporie.core.lexers.**ClassNotFound**

Raised if one of the lookup functions didn't find a matching class.

euporie.core.lexers.**detect_lexer**(*text: str = ''*, *path:* Path | *None = None*, *language: str = ''*) → PygmentsLexerCls | None

Detect the pygments lexer for a file.

### euporie.core.log

Initiate logging for euporie.core.

### Functions

| | |
|---|---|
| *add_log_level*(name, number) | Add a new level to the logger. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *create_output*([stdout, always_prefer_tty]) | Return an `Output` instance for the command line. |
| *dict_merge*(target_dict, input_dict) | Merge the second dictionary onto the first. |
| *get_app_session*() | |
| *get_style_by_name*(name) | Return a style class by its short name. |
| *handle_exception*(exc_type, exc_value, ...) | Log unhandled exceptions and their tracebacks in the log. |
| *indent*(ft[, margin, style, skip_first]) | Indent formatted text with a given margin. |
| *lex*(ft, lexer_name) | Format formatted text using a named `pygments` lexer. |
| *merge_styles*(styles) | Merge multiple *Style* objects. |
| *print_formatted_text*(*values[, sep, end, ...]) | . |
| *renderer_print_formatted_text*(output, ...[, ...]) | Print a list of (style_str, text) tuples in the given style to the output. |
| *setup_logs*([config]) | Configure the logger for euporie. |
| *style_from_pygments_cls*(pygments_style_cls) | Shortcut to create a *Style* instance from a Pygments style class and a style dictionary. |
| *wrap*(ft, width[, style, placeholder, left, ...]) | Wrap formatted text at a given width. |

### euporie.core.log.add_log_level

euporie.core.log.**add_log_level**(*name: str*, *number: int*) → None

Add a new level to the logger.

### euporie.core.log.add_setting

euporie.core.log.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.log.create_output

euporie.core.log.**create_output**(*stdout: TextIO | None = None*, *always_prefer_tty: bool = False*) → Output

Return an `Output` instance for the command line.

**Parameters**

- **stdout** – The stdout object
- **always_prefer_tty** – When set, look for *sys.stderr* if *sys.stdout* is not a TTY. Useful if *sys.stdout* is redirected to a file, but we still want user input and output on the terminal.

  By default, this is *False*. If *sys.stdout* is not a terminal (maybe it's redirected to a file), then a *PlainTextOutput* will be returned. That way, tools like *print_formatted_text* will write plain text into that file.

### euporie.core.log.dict_merge

euporie.core.log.**dict_merge**(*target_dict: dict*, *input_dict: dict*) → None

Merge the second dictionary onto the first.

### euporie.core.log.get_app_session

euporie.core.log.**get_app_session**() → AppSession

### euporie.core.log.get_style_by_name

euporie.core.log.**get_style_by_name**(*name*)

Return a style class by its short name. The names of the builtin styles are listed in `pygments.styles.STYLE_MAP`.

Will raise *pygments.util.ClassNotFound* if no style of that name is found.

### euporie.core.log.handle_exception

euporie.core.log.**handle_exception**(*exc_type: type[BaseException]*, *exc_value: BaseException*, *exc_traceback: TracebackType | None*) → Any

Log unhandled exceptions and their tracebacks in the log.

**Parameters**

- **exc_type** – The type of the exception
- **exc_value** – The exception instance
- **exc_traceback** – The associated traceback

### euporie.core.log.indent

euporie.core.log.**indent**(*ft: StyleAndTextTuples*, *margin: str = ' '*, *style: str = ''*, *skip_first: bool = False*) → StyleAndTextTuples

Indent formatted text with a given margin.

**Parameters**

- **ft** – The formatted text to strip
- **margin** – The margin string to add
- **style** – The style to apply to the margin
- **skip_first** – If `True`, the first line is skipped

**Returns**

The indented formatted text

### euporie.core.log.lex

euporie.core.log.**lex**(*ft: StyleAndTextTuples*, *lexer_name: str*) → StyleAndTextTuples

Format formatted text using a named `pygments` lexer.

### euporie.core.log.merge_styles

euporie.core.log.**merge_styles**(*styles: list[*prompt_toolkit.styles.base.BaseStyle*]*) → _MergedStyle

Merge multiple *Style* objects.

### euporie.core.log.print_formatted_text

euporie.core.log.**print_formatted_text**(**values: Any*, *sep: str = ' '*, *end: str = '\n'*, *file:* TextIO | *None = None*, *flush: bool = False*, *style:* prompt_toolkit.styles.base.BaseStyle | *None = None*, *output: prompt_toolkit.output.base.Output* | *None = None*, *color_depth:* prompt_toolkit.output.color_depth.ColorDepth | *None = None*, *style_transformation: prompt_toolkit.styles.style_transformation.StyleTransformation* | *None = None*, *include_default_pygments_style: bool = True*) → None

```
print_formatted_text(*values, sep=' ', end='\n', file=None, flush=False,␣
↪style=None, output=None)
```

Print text to stdout. This is supposed to be compatible with Python's print function, but supports printing of formatted text. You can pass a `FormattedText`, `HTML` or `ANSI` object to print formatted text.

- Print HTML as follows:

```
print_formatted_text(HTML('<i>Some italic text</i> <ansired>This is red!</
↪ansired>'))

style = Style.from_dict({
    'hello': '#ff0066',
    'world': '#884444 italic',
})
print_formatted_text(HTML('<hello>Hello</hello> <world>world</world>!'),␣
↪style=style)
```

- Print a list of (style_str, text) tuples in the given style to the output. E.g.:

```
style = Style.from_dict({
    'hello': '#ff0066',
    'world': '#884444 italic',
})
fragments = FormattedText([
    ('class:hello', 'Hello'),
    ('class:world', 'World'),
])
print_formatted_text(fragments, style=style)
```

If you want to print a list of Pygments tokens, wrap it in `PygmentsTokens` to do the conversion.

If a prompt_toolkit *Application* is currently running, this will always print above the application or prompt (similar to *patch_stdout*). So, *print_formatted_text* will erase the current application, print the text, and render the application again.

**Parameters**

- **values** – Any kind of printable object, or formatted string.

- **sep** – String inserted between values, default a space.
- **end** – String appended after the last value, default a newline.
- **style** – *Style* instance for the color scheme.
- **include_default_pygments_style** – *bool*. Include the default Pygments style when set to *True* (the default).

### euporie.core.log.renderer_print_formatted_text

euporie.core.log.**renderer_print_formatted_text**(*output: Output*, *formatted_text: AnyFormattedText*, *style:* BaseStyle, *style_transformation: StyleTransformation* | *None* = *None*, *color_depth:* ColorDepth | *None* = *None*) → None

Print a list of (style_str, text) tuples in the given style to the output.

### euporie.core.log.setup_logs

euporie.core.log.**setup_logs**(*config:* Config | *None* = *None*) → None

Configure the logger for euporie.

### euporie.core.log.style_from_pygments_cls

euporie.core.log.**style_from_pygments_cls**(*pygments_style_cls: type[PygmentsStyle]*) → *Style*

Shortcut to create a *Style* instance from a Pygments style class and a style dictionary.

Example:

```
from prompt_toolkit.styles.from_pygments import style_from_pygments_cls
from pygments.styles import get_style_by_name
style = style_from_pygments_cls(get_style_by_name('monokai'))
```

**Parameters**

**pygments_style_cls** – Pygments style class to start from.

### euporie.core.log.wrap

euporie.core.log.**wrap**(*ft: StyleAndTextTuples*, *width: int*, *style: str* = '', *placeholder: str* = '…', *left: int* = *0*, *truncate_long_words: bool* = *True*, *strip_trailing_ws: bool* = *False*, *margin: str* = '') → StyleAndTextTuples

Wrap formatted text at a given width.

If words are longer than the given line they will be truncated

**Parameters**

- **ft** – The formatted text to wrap
- **width** – The width at which to wrap the text
- **style** – The style to apply to the truncation placeholder

- **placeholder** – The string that will appear at the end of a truncated line
- **left** – The starting position within the first line
- **truncate_long_words** – If `True` words longer than a line will be truncated
- **strip_trailing_ws** – If `True`, trailing whitespace will be removed from the ends of lines
- **margin** – Text to use a margin for the continuation of wrapped lines

**Returns**
The wrapped formatted text

## Classes

| | |
|---|---|
| *FormattedText*([iterable]) | A list of `(style, text)` tuples. |
| *FormattedTextHandler*([stream, share_stream, ...]) | Format log records for display on the standard output. |
| *FtFormatter*(*args, **kwargs) | Base class for formatted text logging formatter. |
| *LogTabFormatter*(*args, **kwargs) | Format log messages for display in the log view tab. |
| *Path*(*args, **kwargs) | PurePath subclass that can make system calls. |
| *PseudoTTY*(underlying[, isatty]) | Make an output stream look like a TTY. |
| *QueueHandler*(*args, queue[, style]) | This handler store logs events into a queue. |
| *StdoutFormatter*(*args, **kwargs) | A log formatter for formatting log entries for display on the standard output. |
| *StringIO*([initial_value, newline]) | Text I/O implementation using an in-memory buffer. |
| *Style*(style_rules) | Create a `Style` instance from a list of style rules. |
| *deque* | deque([iterable[, maxlen]]) --> deque object |
| *stdout_to_log*(log[, output]) | A decorator which captures standard output and logs it. |

### euporie.core.log.FormattedText

**class** euporie.core.log.**FormattedText**(*iterable=()*, */*)

A list of `(style, text)` tuples.

(In some situations, this can also be `(style, text, mouse_handler)` tuples.)

### euporie.core.log.FormattedTextHandler

**class** euporie.core.log.**FormattedTextHandler**(*stream: str | TextIO | PseudoTTY | None = None*, *share_stream: bool = True*, *style:* BaseStyle *| None = None*, *pygments_theme: str = 'euporie'*)

Format log records for display on the standard output.

### euporie.core.log.FtFormatter

**class** euporie.core.log.**FtFormatter**(**args: Any*, ***kwargs: Any*)

Base class for formatted text logging formatter.

### euporie.core.log.LogTabFormatter

**class** euporie.core.log.**LogTabFormatter**(**args: Any*, ***kwargs: Any*)

Format log messages for display in the log view tab.

### euporie.core.log.Path

**class** euporie.core.log.**Path**(**args*, ***kwargs*)

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

### euporie.core.log.PseudoTTY

**class** euporie.core.log.**PseudoTTY**(*underlying: IO[str] | TextIO*, *isatty: bool = True*)

Make an output stream look like a TTY.

### euporie.core.log.QueueHandler

**class** euporie.core.log.**QueueHandler**(**args: Any*, *queue:* deque, *style:* Style | *None = None*, ***kwargs: Any*)

This handler store logs events into a queue.

### euporie.core.log.StdoutFormatter

**class** euporie.core.log.**StdoutFormatter**(**args: Any*, ***kwargs: Any*)

A log formatter for formatting log entries for display on the standard output.

### euporie.core.log.StringIO

**class** euporie.core.log.**StringIO**(*initial_value=''*, *newline='\n'*)

Text I/O implementation using an in-memory buffer.

The initial_value argument sets the value of object. The newline argument is like the one of TextIOWrapper's constructor.

### euporie.core.log.Style

**class** euporie.core.log.**Style**(*style_rules: list[tuple[str, str]]*)

Create a `Style` instance from a list of style rules.

The *style_rules* is supposed to be a list of ('classnames', 'style') tuples. The classnames are a whitespace separated string of class names and the style string is just like a Pygments style definition, but with a few additions: it supports 'reverse' and 'blink'.

Later rules always override previous rules.

Usage:

```
Style([
    ('title', '#ff0000 bold underline'),
    ('something-else', 'reverse'),
    ('class1 class2', 'reverse'),
])
```

The `from_dict` classmethod is similar, but takes a dictionary as input.

### euporie.core.log.deque

**class** euporie.core.log.**deque**

deque([iterable[, maxlen]]) –> deque object

A list-like sequence optimized for data accesses near its endpoints.

### euporie.core.log.stdout_to_log

**class** euporie.core.log.**stdout_to_log**(*log: Logger*, *output: str = "Literal['stdout','stderr']"*)

A decorator which captures standard output and logs it.

**class** euporie.core.log.**FormattedTextHandler**(*stream: str | TextIO | PseudoTTY | None = None*, *share_stream: bool = True*, *style:* BaseStyle *| None = None*, *pygments_theme: str = 'euporie'*)

Bases: `StreamHandler`

Format log records for display on the standard output.

**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: LogRecord*) → None

Emit a formatted record.

**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 by returning a false value. If a filter attached to a handler returns a log record instance, then that instance is used in place of the original log record in any further processing of the event by that handler. If a filter returns any other true value, the original log record is used in any further processing of the event by that handler.

If none of the filters return false values, this method returns a log record. If any of the filters return a false value, this method returns a false value.

Changed in version 3.2: Allow filters to be just callables.

Changed in version 3.12: Allow filters to return a LogRecord instead of modifying it in place.

**flush**()

Flushes the stream.

**format**(*record*)

Format the specified record.

If a formatter is set, use it. Otherwise, use the default formatter for the module.

**formatter: *FtFormatter***

**ft_format**(*record: LogRecord*) → *FormattedText*

Format the specified record.

**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 an instance of the log record that was emitted if it passed all filters, otherwise a false value is returned.

**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.

**property name**

**release**()

Release the I/O thread lock.

**removeFilter**(*filter*)

Remove the specified filter from this handler.

**setFormatter**(*fmt*)

Set the formatter for this handler.

**setLevel**(*level*)

Set the logging level of this handler. level must be an int or a str.

**setStream**(*stream*)

Sets the StreamHandler's stream to the specified value, if it is different.

Returns the old stream, if the stream was changed, or None if it wasn't.

**set_name**(*name*)

**terminator = '\n'**

**class** euporie.core.log.**FtFormatter**(**args: Any*, ***kwargs: Any*)

Bases: Formatter

Base class for formatted text logging formatter.

**converter**()

**localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,**
tm_sec,tm_wday,tm_yday,tm_isdst)

Convert seconds since the Epoch to a time tuple expressing local time. When 'seconds' is not passed in, convert the current time instead.

**default_msec_format = '%s,%03d'**

**default_time_format = '%Y-%m-%d %H:%M:%S'**

**format**(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

**formatException**(*ei*)

Format and return the specified exception information as a string.

This default implementation just uses traceback.print_exception()

**formatMessage**(*record*)

**formatStack**(*stack_info*)

This method is provided as an extension point for specialized formatting of stack information.

The input data is a string as returned from a call to traceback.print_stack(), but with the last trailing newline removed.

The base implementation just returns the value passed in.

**formatTime**(*record*, *datefmt=None*)

Return the creation time of the specified LogRecord as formatted text.

This method should be called from format() by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if datefmt (a string) is specified, it is used with time.strftime() to format the creation time of the record. Otherwise, an ISO8601-like (or RFC 3339-like) format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, time.localtime()

is used; to change this for a particular formatter instance, set the 'converter' attribute to a function with the same signature as time.localtime() or time.gmtime(). To change it for all formatters, for example if you want all logging times to be shown in GMT, set the 'converter' attribute in the Formatter class.

**format_traceback**(*tb: str*) → StyleAndTextTuples

Format a traceback string using pygments.

**ft_format**(*record: LogRecord*, *width: int | None = None*) → *FormattedText*

Format a log record as *FormattedText*.

**prepare**(*record: LogRecord*, *width: int | None = None*) → LogRecord

Format certain attributes on the log record.

**usesTime**()

Check if the format uses the creation time of the record.

**class** euporie.core.log.**LogTabFormatter**(**args: Any*, ***kwargs: Any*)

Bases: *FtFormatter*

Format log messages for display in the log view tab.

**converter**()

**localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,**
tm_sec,tm_wday,tm_yday,tm_isdst)

Convert seconds since the Epoch to a time tuple expressing local time. When 'seconds' is not passed in, convert the current time instead.

**default_msec_format = '%s,%03d'**

**default_time_format = '%Y-%m-%d %H:%M:%S'**

**format**(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

**formatException**(*ei*)

Format and return the specified exception information as a string.

This default implementation just uses traceback.print_exception()

**formatMessage**(*record*)

**formatStack**(*stack_info*)

This method is provided as an extension point for specialized formatting of stack information.

The input data is a string as returned from a call to `traceback.print_stack()`, but with the last trailing newline removed.

The base implementation just returns the value passed in.

**formatTime**(*record*, *datefmt=None*)

Return the creation time of the specified LogRecord as formatted text.

This method should be called from format() by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if datefmt (a string) is specified, it is used with time.strftime() to format the creation time of the record. Otherwise, an ISO8601-like (or RFC 3339-like) format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, time.localtime() is used; to change this for a particular formatter instance, set the 'converter' attribute to a function with the same signature as time.localtime() or time.gmtime(). To change it for all formatters, for example if you want all logging times to be shown in GMT, set the 'converter' attribute in the Formatter class.

**format_traceback**(*tb: str*) → StyleAndTextTuples

Format a traceback string using pygments.

**ft_format**(*record: LogRecord*, *width: int | None = None*) → *FormattedText*

Format a log record as formatted text.

**prepare**(*record: LogRecord*, *width: int | None = None*) → LogRecord

Format certain attributes on the log record.

**usesTime**()

Check if the format uses the creation time of the record.

**class** euporie.core.log.**QueueHandler**(**args: Any*, *queue:* deque, *style:* Style | *None = None*, ***kwargs: Any*)

Bases: 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: 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 by returning a false value. If a filter attached to a handler returns a log record instance, then that instance is used in place of the original log record in any further processing of the event by that handler. If a filter returns any other true value, the original log record is used in any further processing of the event by that handler.

If none of the filters return false values, this method returns a log record. If any of the filters return a false value, this method returns a false value.

Changed in version 3.2: Allow filters to be just callables.

Changed in version 3.12: Allow filters to return a LogRecord instead of modifying it in place.

**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.

**formatter: *FtFormatter***

**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 an instance of the log record that was emitted if it passed all filters, otherwise a false value is returned.

**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

Add a hook to run after each log entry.

**Parameters**
**hook** – The hook function to add

**Returns**
The hook id

**hook_id = 0**

**hooks: ClassVar[dict[int, Callable]] = {}**

**property name**

**release**()

Release the I/O thread lock.

**removeFilter**(*filter*)

Remove the specified filter from this handler.

**setFormatter**(*fmt*)

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

Remove a hook function.

**Parameters**
**hook_id** – The ID of the hook function to remove

**class** euporie.core.log.**StdoutFormatter**(**args: Any*, ***kwargs: Any*)

Bases: *FtFormatter*

A log formatter for formatting log entries for display on the standard output.

**converter**()

**localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,**
tm_sec,tm_wday,tm_yday,tm_isdst)

Convert seconds since the Epoch to a time tuple expressing local time. When 'seconds' is not passed in, convert the current time instead.

**default_msec_format = '%s,%03d'**

**default_time_format = '%Y-%m-%d %H:%M:%S'**

**format**(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

**formatException**(*ei*)

Format and return the specified exception information as a string.

This default implementation just uses traceback.print_exception()

**formatMessage**(*record*)

**formatStack**(*stack_info*)

This method is provided as an extension point for specialized formatting of stack information.

The input data is a string as returned from a call to traceback.print_stack(), but with the last trailing newline removed.

The base implementation just returns the value passed in.

**formatTime**(*record*, *datefmt=None*)

Return the creation time of the specified LogRecord as formatted text.

This method should be called from format() by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if datefmt (a string) is specified, it is used with time.strftime() to format the creation time of the record. Otherwise, an ISO8601-like (or RFC 3339-like) format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, time.localtime() is used; to change this for a particular formatter instance, set the 'converter' attribute to a function with the same signature as time.localtime() or time.gmtime(). To change it for all formatters, for example if you want all logging times to be shown in GMT, set the 'converter' attribute in the Formatter class.

**format_traceback**(*tb: str*) → StyleAndTextTuples

Format a traceback string using pygments.

**ft_format**(*record: LogRecord*, *width: int | None = None*) → *FormattedText*

Format log records for display on the standard output.

**prepare**(*record: LogRecord*, *width: int | None = None*) → LogRecord

Format certain attributes on the log record.

**usesTime**()

Check if the format uses the creation time of the record.

euporie.core.log.**add_log_level**(*name: str*, *number: int*) → None

Add a new level to the logger.

euporie.core.log.**handle_exception**(*exc_type: type[BaseException]*, *exc_value: BaseException*, *exc_traceback: TracebackType | None*) → Any

Log unhandled exceptions and their tracebacks in the log.

**Parameters**

- **exc_type** – The type of the exception
- **exc_value** – The exception instance
- **exc_traceback** – The associated traceback

euporie.core.log.**setup_logs**(*config: Config | None = None*) → None

Configure the logger for euporie.

**class** euporie.core.log.**stdout_to_log**(*log: Logger*, *output: str = "Literal['stdout','stderr']"*)

Bases: object

A decorator which captures standard output and logs it.

### euporie.core.lsp

Defines a simple LSP client.

### Functions

| | |
|---|---|
| *NamedTuple*(typename[, fields]) | Typed version of namedtuple. |
| *range_to_slice*(start_line, start_char, ...) | Convert a line/character ranger to a slice. |

### euporie.core.lsp.NamedTuple

euporie.core.lsp.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)

Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

### euporie.core.lsp.range_to_slice

euporie.core.lsp.**range_to_slice**(*start_line: int*, *start_char: int*, *end_line: int*, *end_char: int*, *text: str*) → slice

Convert a line/character ranger to a slice.

### Classes

| | |
|---|---|
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *LspCell*(id, idx, path, kind, language, text, ...) | An LSP client's representation of a cell. |
| *LspClient*(name, command[, languages, settings]) | A client for communicating with LSP servers. |
| *Path*(*args, **kwargs) | PurePath subclass that can make system calls. |

### euporie.core.lsp.Event

**class** euporie.core.lsp.**Event**(*sender: _Sender*, *handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.lsp.LspCell

**class** euporie.core.lsp.**LspCell**(*id: str*, *idx: int*, *path:* Path, *kind: str*, *language: str*, *text: str*, *execution_count: int*, *metadata: dict[str, Any] | None = None*)

An LSP client's representation of a cell.

### euporie.core.lsp.LspClient

**class** euporie.core.lsp.**LspClient**(*name: str*, *command: str*, *languages: Sequence[str] | None = None*, *settings: dict | None = None*)

A client for communicating with LSP servers.

### euporie.core.lsp.Path

**class** euporie.core.lsp.**Path**(**args*, ***kwargs*)

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

**class** euporie.core.lsp.**LspCell**(*id: str*, *idx: int*, *path:* Path, *kind: str*, *language: str*, *text: str*, *execution_count: int*, *metadata: dict[str, Any] | None = None*)

Bases: NamedTuple

An LSP client's representation of a cell.

**count**(*value*, */*)

Return number of occurrences of value.

**execution_count: int**

Alias for field number 6

**id: str**

Alias for field number 0

**idx: int**

Alias for field number 1

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**kind: str**

Alias for field number 3

**language: str**

Alias for field number 4

**metadata: dict[str, Any] | None**

Alias for field number 7

**path**: *Path*
Alias for field number 2

**text**: *str*
Alias for field number 5

**class** euporie.core.lsp.**LspClient**(*name: str*, *command: str*, *languages: Sequence[str] | None = None*, *settings: dict | None = None*)

Bases: object

A client for communicating with LSP servers.

**change_doc**(*path:* Path, *language: str*, *text: str*, *debounce: bool = True*) → None
Tell the server we have changed a document file.

**change_nb_add**(*path:* Path, *cells: Sequence[*LspCell*]*) → None
Notify server of cells added to notebook.

**change_nb_delete**(*path:* Path, *cells: Sequence[*LspCell*]*) → None
Notify server of cells removed from a notebook.

**change_nb_edit**(*path:* Path, *cells: Sequence[*LspCell*]*, *debounce: bool = True*) → None
Notify server of a change in notebook cells.

**change_nb_meta**(*path:* Path, *metadata: dict[str, Any]*) → None
Notify server of a change in a notebook metadata.

**close_doc**(*path:* Path) → None
Tell the server we have closed a document file.

**close_nb**(*path:* Path, *cells: Sequence[*LspCell*]*) → None
Tell the server we closed a notebook document.

**complete**(*path:* Path, *line: int*, *char: int*, *timeout: int = 5*) → list[dict[str, str]]
Trigger a LSP completion request.

**async complete_**(*path:* Path, *line: int*, *char: int*, *timeout: int = 5*) → list[dict[str, Any]]
Request a completion from the LSP server.

**diagnostics**(*params: dict[str, Any]*) → None
Receive a diagnstics report from the server.

**exit**() → Future
Tell the server to exit.

**async exit_**(*timeout: int = 1*) → None
Tell the server to exit.

**format**(*path:* Path, *tab_size: int = 4*, *spaces: bool = True*, *timeout: int = 1*) → list[dict] | None
Request LSP document formatting.

**async format_**(*path:* Path, *tab_size: int = 4*, *spaces: bool = True*, *timeout: int = 5*) → list[dict] | None
Trigger a formatting request from the LSP.

**hover**(*path:* Path, *line: int*, *char: int*, *timeout: int = 1*) → dict
Trigger a LSP hover request.

**async hover_**(*path:* Path, *line: int*, *char: int*, *timeout: int = 5*) → dict
Request hover text from the LSP server.

**async initialize**(*root:* pathlib.Path | *None* = *None*) → None

Initiate the LSP server.

**log_message**(*params: dict[str, Any]*) → None

Send a log message from the server to the log.

**loop = <_UnixSelectorEventLoop running=False closed=False debug=False>**

**property next_msg_id: int**

Get and increment the RPC message ID.

**open_doc**(*path:* Path, *language: str*, *text: str*) → None

Tell the server we have opened a document file.

**open_nb**(*path:* Path, *cells: Sequence[*LspCell*]*, *metadata: dict[str, Any]* | *None*) → None

Tell the server we have opened a notebook file.

**async process_msg**(*data: dict*) → None

Process an incoming message from the LSP server.

**save_doc**(*path:* Path, *text: str*) → None

Tell the server we saved a text document.

**save_nb**(*path:* Path) → None

Tell the server we saved a notebook.

**async send_msg**(*method: str*, *params: dict[str, Any]* | *None* = *None*, *cb: Callable[[Any], Coroutine[Any, Any, None]]* | *None* = *None*) → int

Send a message to the LSP server, returning the sent message's ID.

**signature**(*path:* Path, *line: int*, *char: int*, *timeout: int* = *1*) → dict[str, object] | None

Trigger a LSP signature request.

**async signature_**(*path:* Path, *line: int*, *char: int*, *timeout: int* = *5*) → dict[str, object] | None

Request a signature from the LSP server.

**start**(*root:* pathlib.Path | *None* = *None*) → None

Start the LSP server.

**async start_**(*root:* pathlib.Path | *None* = *None*) → None

Launch the LSP server subprocess.

**thread = <Thread(Thread-1 (_setup_loop), initial daemon)>**

**will_save_doc**(*path:* Path) → None

Tell the server we will save a document file.

**will_save_nb**(*path:* Path) → None

Tell the server we will save a notebook file.

euporie.core.lsp.**range_to_slice**(*start_line: int*, *start_char: int*, *end_line: int*, *end_char: int*, *text: str*) → slice

Convert a line/character ranger to a slice.

### euporie.core.margins

Contain margins.

### Functions

| | |
|---|---|
| *cast*(typ, val) | Cast a value to a type. |
| *get_app*() | Get the current active (running) Application. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |

### euporie.core.margins.cast

euporie.core.margins.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.margins.get_app

euporie.core.margins.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.margins.to_filter

euporie.core.margins.**to_filter**(*bool_or_filter: Union[*Filter, *bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *BorderMargin*([char, style]) | A margin which shows a fixed character. |
| *ClickableMargin*() | A margin sub-class which handles mouse events. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *Margin*() | Base interface for a margin. |
| *MarginContainer*(margin, target) | A container which renders a stand-alone margin. |
| *MouseButton*(value[, names, module, ...]) | |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, which also store relative position of the mouse event in a cell. |
| *MouseEventType*(value[, names, module, ...]) | |
| *NumberedMargin*([diagnostics, show_diagnostics]) | Margin that displays the line numbers of a *Window*. |
| *OverflowMargin*() | A margin which indicates lines extending beyond the edge of the window. |
| *Point*(x, y) | |
| *PtkMouseEvent* | alias of *MouseEvent* |
| *RelativePosition*(x, y) | Store the relative position or the mouse within a terminal cell. |
| *ScrollbarMargin*([display_arrows, ...]) | Margin displaying a scrollbar. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |

### euporie.core.margins.ABCMeta

**class** euporie.core.margins.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.margins.BorderMargin

**class** euporie.core.margins.**BorderMargin**(*char: str = '?'*, *style: str = ''*)

A margin which shows a fixed character.

### euporie.core.margins.ClickableMargin

**class** euporie.core.margins.**ClickableMargin**

A margin sub-class which handles mouse events.

### euporie.core.margins.Dimension

**class** euporie.core.margins.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.margins.FormattedTextControl

**class** euporie.core.margins.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or `Filter`: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **key_bindings** – a `KeyBindings` object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.margins.Margin

**class** euporie.core.margins.**Margin**

Base interface for a margin.

### euporie.core.margins.MarginContainer

**class** euporie.core.margins.**MarginContainer**(*margin:* Margin, *target: ScrollableContainer*)

A container which renders a stand-alone margin.

### euporie.core.margins.MouseButton

**class** euporie.core.margins.**MouseButton**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.margins.MouseEvent

**class** euporie.core.margins.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers:* *frozenset*[prompt_toolkit.mouse_events.MouseModifier], *cell_position:* euporie.core.key_binding.bindings.mouse.RelativePosition | *None*)

Mouse event, which also store relative position of the mouse event in a cell.

### euporie.core.margins.MouseEventType

**class** euporie.core.margins.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.margins.NumberedMargin

**class** euporie.core.margins.**NumberedMargin**(*diagnostics: Report | Callable[[], Report] | None = None*, *show_diagnostics: FilterOrBool = False*)

Margin that displays the line numbers of a *Window*.

### euporie.core.margins.OverflowMargin

**class** euporie.core.margins.**OverflowMargin**

A margin which indicates lines extending beyond the edge of the window.

### euporie.core.margins.Point

**class** euporie.core.margins.**Point**(*x*, *y*)

### euporie.core.margins.PtkMouseEvent

euporie.core.margins.**PtkMouseEvent**

alias of *MouseEvent*

### euporie.core.margins.RelativePosition

**class** euporie.core.margins.**RelativePosition**(*x: float*, *y: float*)

Store the relative position or the mouse within a terminal cell.

### euporie.core.margins.ScrollbarMargin

**class** euporie.core.margins.**ScrollbarMargin**(*display_arrows: Union[Filter, bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[Filter, bool] = False*, *smooth: bool = True*, *style: str = ''*)

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

### euporie.core.margins.Window

**class** euporie.core.margins.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

**class** euporie.core.margins.**BorderMargin**(*char: str = '│'*, *style: str = ''*)

Bases: *Margin*

A margin which shows a fixed character.

**create_margin**(*window_render_info: WindowRenderInfo*, *width: int*, *height: int*) → StyleAndTextTuples

Generate the margin's content.

**get_width**(*get_ui_content: Callable[[], UIContent]*) → int

Return the width of the margin.

**class** euporie.core.margins.**ClickableMargin**

Bases: *Margin*

A margin sub-class which handles mouse events.

**abstract create_margin**(*window_render_info: WindowRenderInfo*, *width: int*, *height: int*) → StyleAndTextTuples

Creates a margin. This should return a list of (style_str, text) tuples.

**Parameters**

- **window_render_info** – *WindowRenderInfo* instance, generated after rendering and copying the visible part of the *UIControl* into the *Window*.
- **width** – The width that's available for this margin. (As reported by *get_width()*.)
- **height** – The height that's available for this margin. (The height of the *Window*.)

**abstract get_width**(*get_ui_content: Callable[[], UIContent]*) → int

Return the width that this margin is going to consume.

**Parameters**

**get_ui_content** – Callable that asks the user control to create a *UIContent* instance. This can be used for instance to obtain the number of lines.

**set_write_position**(*write_position:* WritePosition) → None

Set the write position of the menu.

**write_position: *WritePosition* | None**

**class** euporie.core.margins.**MarginContainer**(*margin:* Margin, *target: ScrollableContainer*)

Bases: *Window*

A container which renders a stand-alone margin.

**create_fragments**() → StyleAndTextTuples

Generate text fragments to display.

**get_children**() → list[*Container*]

Return the list of child *Container* objects.

**get_key_bindings**() → *KeyBindingsBase* | None

Return a *KeyBindings* object.

**is_modal**() → bool

When this container is modal.

**preferred_height**(*width: int*, *max_available_height: int*) → *Dimension*

Return a thedesired height for this container.

**preferred_width**(*max_available_width: int*) → *Dimension*

Return a the desired width for this container.

**reset**() → None

Reset the state of this container and all the children.

**write_to_screen**(*screen:* Screen, *mouse_handlers:* MouseHandlers, *write_position:* WritePosition, *parent_style: str*, *erase_bg: bool*, *z_index: int* | *None*) → None

Write the actual content to the screen.

**class** euporie.core.margins.**NumberedMargin**(*diagnostics:* Report | *Callable[[],* Report*]* | *None* = *None*, *show_diagnostics: FilterOrBool = False*)

Bases: *Margin*

Margin that displays the line numbers of a *Window*.

**create_margin**(*window_render_info:* WindowRenderInfo, *width: int*, *height: int*) → StyleAndTextTuples

Generate the margin's content.

**get_width**(*get_ui_content: Callable[[],* UIContent*]*) → int

Return the width of the margin.

**style = 'class:line-number'**

**class** euporie.core.margins.**OverflowMargin**

Bases: *Margin*

A margin which indicates lines extending beyond the edge of the window.

**create_margin**(*window_render_info:* WindowRenderInfo, *width: int*, *height: int*) → StyleAndTextTuples

Generate the margin's content.

**get_width**(*get_ui_content: Callable[[],* UIContent*]*) → int

Return the width of the margin.

**class** euporie.core.margins.**ScrollbarMargin**(*display_arrows: Union[Filter, bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[Filter, bool] = False*, *smooth: bool = True*, *style: str = ''*)

Bases: *ClickableMargin*

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 | None*, *width: int*, *height: int*, *margin_render_info: WindowRenderInfo | None = None*) → StyleAndTextTuples

Create the margin's formatted text.

**eighths** = '▁▂▃▄▅▆▇ '

**get_width**(*get_ui_content: Callable[[], UIContent]*) → int

Return the scrollbar width: always 1.

**mouse_handler**(*mouse_event: PtkMouseEvent*) → NotImplementedOrNone

Type compatible mouse handler.

**async repeat**(*mouse_event:* MouseEvent, *timeout: float = 0.1*) → None

Repeat a mouse event after a timeout.

**set_write_position**(*write_position:* WritePosition) → None

Set the write position of the menu.

**write_position**: *WritePosition* | **None**

## euporie.core.path

Responsible for loading data from urls.

### Functions

| | |
|---|---|
| *fs_register_implementation*(name, cls[, ...]) | Add implementation class to the registry |
| *parse_path*(path[, resolve]) | Parse and resolve a path. |

### euporie.core.path.fs_register_implementation

euporie.core.path.**fs_register_implementation**(*name*, *cls*, *clobber=False*, *errtxt=None*)

Add implementation class to the registry

**Parameters**

- **name** (*str*) – Protocol name to associate with the class
- **cls** (*class or str*) – if a class: fsspec-compliant implementation class (normally inherits from `fsspec.AbstractFileSystem`, gets added straight to the registry. If a str, the full path to an implementation class like package.module.class, which gets added to known_implementations, so the import is deferred until the filesystem is actually used.
- **clobber** (*bool (optional)*) – Whether to overwrite a protocol with the same name; if False, will raise instead.
- **errtxt** (*str (optional)*) – If given, then a failure to import the given class will result in this text being given.

### euporie.core.path.parse_path

euporie.core.path.**parse_path**(*path: str | PathLike*, *resolve: bool = True*) → *Path*

Parse and resolve a path.

### Classes

| | |
|---|---|
| *ClientResponse*(method, url, *, writer, ...) | |
| *FsHTTPFileSystem* | alias of `HTTPFileSystem` |
| *HTTPFileSystem*(*args, **kwargs) | A *HTTPFileSystem* which does not raise exceptions on 404 errors. |
| *Path*(*args, **kwargs) | PurePath subclass that can make system calls. |
| *UPath*(*args[, protocol]) | |
| *UntitledPath*(*args[, protocol]) | A path for untitled files, as needed for LSP servers. |

### euporie.core.path.ClientResponse

**class** euporie.core.path.**ClientResponse**(*method: str*, *url: URL*, *, *writer: asyncio.Task[None]*, *continue100: Optional[asyncio.Future[bool]]*, *timer: BaseTimerContext*, *request_info: RequestInfo*, *traces: List[Trace]*, *loop: AbstractEventLoop*, *session: ClientSession*)

### euporie.core.path.FsHTTPFileSystem

euporie.core.path.**FsHTTPFileSystem**

alias of HTTPFileSystem

### euporie.core.path.HTTPFileSystem

**class** euporie.core.path.**HTTPFileSystem**(**args*, ***kwargs*)

A *HTTPFileSystem* which does not raise exceptions on 404 errors.

### euporie.core.path.Path

**class** euporie.core.path.**Path**(**args*, ***kwargs*)

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

### euporie.core.path.UPath

**class** euporie.core.path.**UPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

### euporie.core.path.UntitledPath

**class** euporie.core.path.**UntitledPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

A path for untitled files, as needed for LSP servers.

**class** euporie.core.path.**HTTPFileSystem**(**args*, ***kwargs*)

Bases: HTTPFileSystem

A *HTTPFileSystem* which does not raise exceptions on 404 errors.

**async_impl = True**

**blocksize = 4194304**

**cachable = True**

**cat**(*path*, *recursive=False*, *on_error='raise'*, ***kwargs*)

Fetch (potentially multiple) paths' contents

**Parameters**

- **recursive** (*bool*) – If True, assume the path(s) are directories, and get all the contained files
- **on_error** (*"raise", "omit", "return"*) – If raise, an underlying exception will be raised (converted to KeyError if the type is in self.missing_exceptions); if omit, keys with exception will simply not be included in the output; if "return", all keys are included in the output, but the value will be bytes or an exception instance.

- **kwargs** (*passed to cat_file*) –

**Returns**

- **dict of {path** (*contents} if there are multiple paths*)
- *or the path has been otherwise expanded*

**cat_file**(*path*, *start=None*, *end=None*, ***kwargs*)

Get the content of a file

**Parameters**

- **path** (*URL of file on this filesystems*) –
- **start** (*int*) – Bytes limits of the read. If negative, backwards from end, like usual python slices. Either can be None for start or end of file, respectively
- **end** (*int*) – Bytes limits of the read. If negative, backwards from end, like usual python slices. Either can be None for start or end of file, respectively
- **kwargs** (passed to `open()`.) –

**cat_ranges**(*paths*, *starts*, *ends*, *max_gap=None*, *on_error='return'*, ***kwargs*)

Get the contents of byte ranges from one or more files

**Parameters**

- **paths** (*list*) – A list of of filepaths on this filesystems
- **starts** (*int or list*) – Bytes limits of the read. If using a single int, the same value will be used to read all the specified files.
- **ends** (*int or list*) – Bytes limits of the read. If using a single int, the same value will be used to read all the specified files.

**checksum**(*path*)

Unique value for current version of file

If the checksum is the same from one moment to another, the contents are guaranteed to be the same. If the checksum changes, the contents *might* have changed.

This should normally be overridden; default will probably capture creation/modification timestamp (which would be good) or maybe access timestamp (which would be bad)

**classmethod clear_instance_cache**()

Clear the cache of filesystem instances.

### Notes

Unless overridden by setting the `cachable` class attribute to False, the filesystem class stores a reference to newly created instances. This prevents Python's normal rules around garbage collection from working, since the instances refcount will not drop to zero until `clear_instance_cache` is called.

**static close_session**(*loop*, *session*)

**copy**(*path1*, *path2*, *recursive=False*, *maxdepth=None*, *on_error=None*, ***kwargs*)

Copy within two locations in the filesystem

**on_error**

["raise", "ignore"] If raise, any not-found exceptions will be raised; if ignore any not-found exceptions will cause the path to be skipped; defaults to raise unless recursive is true, where the default is ignore

**cp**(*path1*, *path2*, ***kwargs*)

Alias of *AbstractFileSystem.copy*.

**cp_file**(*path1*, *path2*, ***kwargs*)

**created**(*path*)

Return the created timestamp of a file as a datetime.datetime

**classmethod current**()

Return the most recently instantiated FileSystem

If no instance has been created, then create one with defaults

**delete**(*path*, *recursive=False*, *maxdepth=None*)

Alias of *AbstractFileSystem.rm*.

**disable_throttling = False**

**disk_usage**(*path*, *total=True*, *maxdepth=None*, ***kwargs*)

Alias of *AbstractFileSystem.du*.

**download**(*rpath*, *lpath*, *recursive=False*, ***kwargs*)

Alias of *AbstractFileSystem.get*.

**du**(*path*, *total=True*, *maxdepth=None*, *withdirs=False*, ***kwargs*)

Space used by files and optionally directories within a path

Directory size does not include the size of its contents.

**Parameters**

- **path** (*str*) –
- **total** (*bool*) – Whether to sum all the file sizes
- **maxdepth** (*int or None*) – Maximum number of directory levels to descend, None for unlimited.
- **withdirs** (*bool*) – Whether to include directory paths in the output.
- **kwargs** (passed to `find`) –

**Returns**

- **Dict of {path** (*size} if total=False, or int otherwise, where numbers*)
- *refer to bytes used.*

**encode_url**(*url*)

**end_transaction**()

Finish write transaction, non-context version

**exists**(*path*, ***kwargs*)

Is there a file at the given path

**expand_path**(*path*, *recursive=False*, *maxdepth=None*, ***kwargs*)

Turn one or more globs or directories into a list of all matching paths to files or directories.

kwargs are passed to `glob` or `find`, which may in turn call `ls`

**find**(*path*, *maxdepth=None*, *withdirs=False*, *detail=False*, ***kwargs*)

List all files below path.

Like posix `find` command without conditions

**Parameters**

- **path** (*str*) –
- **maxdepth** (*int or None*) – If not None, the maximum number of levels to descend
- **withdirs** (*bool*) – Whether to include directory paths in the output. This is True when used by glob, but users usually only want files.
- **ls.** (*kwargs are passed to*) –

**static from_json**(*blob*)

Recreate a filesystem instance from JSON representation

See `.to_json()` for the expected structure of the input

**Parameters**
**blob** (*str*) –

**Return type**
file system instance, not necessarily of this particular class.

**property fsid**

Persistent filesystem id that can be used to compare filesystems across sessions.

**get**(*rpath*, *lpath*, *recursive=False*, *callback=<fsspec.callbacks.NoOpCallback object>*, *maxdepth=None*, ***kwargs*)

Copy file(s) to local.

Copies a specific file or tree of files (if recursive=True). If lpath ends with a "/", it will be assumed to be a directory, and target files will go within. Can submit a list of paths, which may be glob-patterns and will be expanded.

Calls get_file for each source.

**get_file**(*rpath*, *lpath*, *callback=<fsspec.callbacks.NoOpCallback object>*, *outfile=None*, ***kwargs*)

Copy single remote file to local

**get_mapper**(*root=''*, *check=False*, *create=False*, *missing_exceptions=None*)

Create key/value store based on this file-system

Makes a MutableMapping interface to the FS at the given root path. See `fsspec.mapping.FSMap` for further details.

**glob**(*path*, *maxdepth=None*, ***kwargs*)

Find files by glob-matching.

If the path ends with '/', only folders are returned.

We support `"**"`, `"?"` and `"[..]"`. We do not support ^ for pattern negation.

The *maxdepth* option is applied on the first ** found in the path.

kwargs are passed to `ls`.

**head**(*path*, *size=1024*)

Get the first `size` bytes from file

**info** (*path*, ***kwargs*)

Give details of entry at path

Returns a single dictionary, with exactly the same information as `ls` would with `detail=True`.

The default implementation should calls ls and could be overridden by a shortcut. kwargs are passed on to `ls()`.

Some file systems might not be able to measure the file's size, in which case, the returned dict will include `'size': None`.

**Returns**

- **dict with keys** (*name (full path in the FS), size (in bytes), type (file,)*
- *directory, or something else) and other FS-specific keys.*

**invalidate_cache** (*path=None*)

Discard any cached directory information

**Parameters**

**path** (*string or None*) – If None, clear all listings cached else listings at or under given path.

**isdir** (*path*)

Is this entry directory-like?

**isfile** (*path*)

Is this entry file-like?

**lexists** (*path*, ***kwargs*)

If there is a file at the given path (including broken links)

**listdir** (*path*, *detail=True*, ***kwargs*)

Alias of *AbstractFileSystem.ls*.

**property loop**

**ls** (*url*, *detail=True*, ***kwargs*)

List objects at path.

This should include subdirectories and files at that location. The difference between a file and a directory must be clear when details are requested.

The specific keys, or perhaps a FileInfo class, or similar, is TBD, but must be consistent across implementations. Must include:

- full path to the entry (without protocol)
- size of the entry, in bytes. If the value cannot be determined, will be `None`.
- type of entry, "file", "directory" or other

Additional information may be present, appropriate to the file-system, e.g., generation, checksum, etc.

May use refresh=True|False to allow use of self._ls_from_cache to check for a saved listing and avoid calling the backend. This would be common where listing may be expensive.

**Parameters**

- **path** (*str*) –
- **detail** (*bool*) – if True, gives a list of dictionaries, where each is the same as the result of `info(path)`. If False, gives a list of paths (str).

- **kwargs** (*may have additional backend-specific options, such as version*) – information

**Returns**

- *List of strings if detail is False, or list of directory information*
- *dicts if detail is True.*

**makedir**(*path*, *create_parents=True*, ***kwargs*)

Alias of *AbstractFileSystem.mkdir*.

**makedirs**(*path*, *exist_ok=False*)

Recursively make directories

Creates directory at path and any intervening required directories. Raises exception if, for instance, the path already exists but is a file.

**Parameters**

- **path** (*str*) – leaf directory name
- **exist_ok** (*bool (False)*) – If False, will error if the target already exists

**mirror_sync_methods = True**

**mkdir**(*path*, *create_parents=True*, ***kwargs*)

Create directory entry at path

For systems that don't have true directories, may create an for this instance only and not touch the real filesystem

**Parameters**

- **path** (*str*) – location
- **create_parents** (*bool*) – if True, this is equivalent to `makedirs`
- **kwargs** – may be permissions, etc.

**mkdirs**(*path*, *exist_ok=False*)

Alias of *AbstractFileSystem.makedirs*.

**modified**(*path*)

Return the modified timestamp of a file as a datetime.datetime

**move**(*path1*, *path2*, ***kwargs*)

Alias of *AbstractFileSystem.mv*.

**mv**(*path1*, *path2*, *recursive=False*, *maxdepth=None*, ***kwargs*)

Move file(s) from one location to another

**open**(*path*, *mode='rb'*, *block_size=None*, *cache_options=None*, *compression=None*, ***kwargs*)

Return a file-like object from the filesystem

The resultant instance must function correctly in a context `with` block.

**Parameters**

- **path** (*str*) – Target file
- **mode** (*str like 'rb', 'w'*) – See builtin `open()`
- **block_size** (*int*) – Some indication of buffering - this is a value in bytes

- **cache_options** (*dict, optional*) – Extra arguments to pass through to the cache.
- **compression** (*string or None*) – If given, open file using compression codec. Can either be a compression name (a key in `fsspec.compression.compr`) or "infer" to guess the compression from the filename suffix.
- **encoding** (*passed on to TextIOWrapper for text mode*) –
- **errors** (*passed on to TextIOWrapper for text mode*) –
- **newline** (*passed on to TextIOWrapper for text mode*) –

**async open_async**(*path*, *mode='rb'*, *size=None*, ***kwargs*)

**pipe**(*path*, *value=None*, ***kwargs*)

Put value into path

(counterpart to `cat`)

**Parameters**

- **path** (*string or dict(str, bytes)*) – If a string, a single remote location to put `value` bytes; if a dict, a mapping of {path: bytesvalue}.
- **value** (*bytes, optional*) – If using a single path, these are the bytes to put there. Ignored if `path` is a dict

**pipe_file**(*path*, *value*, ***kwargs*)

Set the bytes of given file

**protocol: ClassVar[str | tuple[str, ...]] = 'abstract'**

**put**(*lpath*, *rpath*, *recursive=False*, *callback=<fsspec.callbacks.NoOpCallback object>*, *maxdepth=None*, ***kwargs*)

Copy file(s) from local.

Copies a specific file or tree of files (if recursive=True). If rpath ends with a "/", it will be assumed to be a directory, and target files will go within.

Calls put_file for each source.

**put_file**(*lpath*, *rpath*, *callback=<fsspec.callbacks.NoOpCallback object>*, ***kwargs*)

Copy single file to remote

**read_block**(*fn*, *offset*, *length*, *delimiter=None*)

Read a block of bytes from

Starting at `offset` of the file, read `length` bytes. If `delimiter` is set then we ensure that the read starts and stops at delimiter boundaries that follow the locations `offset` and `offset + length`. If `offset` is zero then we start at zero. The bytestring returned WILL include the end delimiter string.

If offset+length is beyond the eof, reads to eof.

**Parameters**

- **fn** (*string*) – Path to filename
- **offset** (*int*) – Byte offset to start read
- **length** (*int*) – Number of bytes to read. If None, read to end.
- **delimiter** (*bytes (optional)*) – Ensure reading starts and stops at delimiter bytestring

### Examples

```
>>> fs.read_block('data/file.csv', 0, 13)
b'Alice, 100\nBo'
>>> fs.read_block('data/file.csv', 0, 13, delimiter=b'\n')
b'Alice, 100\nBob, 200\n'
```

Use `length=None` to read to the end of the file. >>> fs.read_block('data/file.csv', 0, None, delimiter=b'n') # doctest: +SKIP b'Alice, 100nBob, 200nCharlie, 300'

**See also:**

`fsspec.utils.read_block()`

**read_bytes**(*path*, *start=None*, *end=None*, ***kwargs*)

Alias of *AbstractFileSystem.cat_file*.

**read_text**(*path*, *encoding=None*, *errors=None*, *newline=None*, ***kwargs*)

Get the contents of the file as a string.

**Parameters**

- **path** (*str*) – URL of file on this filesystems
- **encoding** (same as *open*.) –
- **errors** (same as *open*.) –
- **newline** (same as *open*.) –

**rename**(*path1*, *path2*, ***kwargs*)

Alias of *AbstractFileSystem.mv*.

**rm**(*path*, *recursive=False*, *maxdepth=None*)

Delete files.

**Parameters**

- **path** (*str or list of str*) – File(s) to delete.
- **recursive** (*bool*) – If file(s) are directories, recursively delete contents and then also remove the directory
- **maxdepth** (*int or None*) – Depth to pass to walk for finding files to delete, if recursive. If None, there will be no limit and infinite recursion may be possible.

**rm_file**(*path*)

Delete a file

**rmdir**(*path*)

Remove a directory, if empty

**root_marker = ''**

**sep = '/'**

**async set_session**()

**sign**(*path*, *expiration=100*, ***kwargs*)

Create a signed URL representing the given path

Some implementations allow temporary URLs to be generated, as a way of delegating credentials.

**Parameters**

- **path** (*str*) – The path on the filesystem
- **expiration** (*int*) – Number of seconds to enable the URL for (if supported)

**Returns**
**URL** – The signed URL

**Return type**
str

:raises NotImplementedError : if method is not implemented for a filesystem:

**size**(*path*)

Size in bytes of file

**sizes**(*paths*)

Size in bytes of each file in a list of paths

**start_transaction**()

Begin write transaction for deferring files, non-context version

**stat**(*path*, ***kwargs*)

Alias of *AbstractFileSystem.info*.

**tail**(*path*, *size=1024*)

Get the last `size` bytes from file

**to_json**()

JSON representation of this filesystem instance

**Returns**
**str** – protocol (text name of this class's protocol, first one in case of multiple), args (positional args, usually empty), and all other kwargs as their own keys.

**Return type**
JSON structure with keys cls (the python location of this class),

**touch**(*path*, *truncate=True*, ***kwargs*)

Create empty file, or update timestamp

**Parameters**

- **path** (*str*) – file location
- **truncate** (*bool*) – If True, always set file size to 0; if False, update timestamp and leave file unchanged, if backend allows this

**property transaction**

A context within which files are committed together upon exit

Requires the file class to implement *.commit()* and *.discard()* for the normal and exception cases.

**transaction_type**

alias of `Transaction`

**ukey**(*url*)

Unique identifier; assume HTTP files are static, unchanging

**unstrip_protocol**(*name: str*) → str

Format FS-specific path to generic, including protocol

**upload**(*lpath*, *rpath*, *recursive=False*, ***kwargs*)

Alias of *AbstractFileSystem.put*.

**walk**(*path*, *maxdepth=None*, *topdown=True*, *on_error='omit'*, ***kwargs*)

Return all files belows path

List all files, recursing into subdirectories; output is iterator-style, like `os.walk()`. For a simple list of files, `find()` is available.

When topdown is True, the caller can modify the dirnames list in-place (perhaps using del or slice assignment), and walk() will only recurse into the subdirectories whose names remain in dirnames; this can be used to prune the search, impose a specific order of visiting, or even to inform walk() about directories the caller creates or renames before it resumes walk() again. Modifying dirnames when topdown is False has no effect. (see os.walk)

Note that the "files" outputted will include anything that is not a directory, such as links.

**Parameters**

- **path** (*str*) – Root to recurse into
- **maxdepth** (*int*) – Maximum recursion depth. None means limitless, but not recommended on link-based file-systems.
- **topdown** (*bool (True)*) – Whether to walk the directory tree from the top downwards or from the bottom upwards.
- **on_error** (*"omit", "raise", a collable*) – if omit (default), path with exception will simply be empty; If raise, an underlying exception will be raised; if callable, it will be called with a single OSError instance as argument
- **kwargs** (passed to `ls`) –

**write_bytes**(*path*, *value*, ***kwargs*)

Alias of *AbstractFileSystem.pipe_file*.

**write_text**(*path*, *value*, *encoding=None*, *errors=None*, *newline=None*, ***kwargs*)

Write the text to the given file.

An existing file will be overwritten.

**Parameters**

- **path** (*str*) – URL of file on this filesystems
- **value** (*str*) – Text to write.
- **encoding** (same as *open*.) –
- **errors** (same as *open*.) –
- **newline** (same as *open*.) –

**class** euporie.core.path.**UntitledPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

Bases: *UPath*

A path for untitled files, as needed for LSP servers.

**absolute**()

Return an absolute version of this path by prepending the current working directory. No normalization or symlink resolution is performed.

Use resolve() to get the canonical path to a file.

**property anchor**

The concatenation of the drive and root, or ''.

**as_posix**()

Return the string representation of the path with forward (/) slashes.

**as_uri**()

Return the path as a 'file' URI.

**chmod**(*mode*, *, *follow_symlinks=True*)

Change the permissions of the path, like os.chmod().

**classmethod cwd**()

Return a new path pointing to the current working directory.

**property drive**

The drive prefix (letter or UNC path), if any.

**exists**(*, *follow_symlinks: bool = True*) → bool

Untitled files are unsaved and do not exist.

**expanduser**()

Return a new path with expanded ~ and ~user constructs (as returned by os.path.expanduser)

**property fs: AbstractFileSystem**

The cached fsspec filesystem instance for the path.

**glob**(*pattern: str*, *, *case_sensitive=None*)

Iterate over this subtree and yield all existing files (of any kind, including directories) matching the given relative pattern.

**group**()

Return the group name of the file gid.

**hardlink_to**(*target*)

Make this path a hard link pointing to the same file as *target*.

Note the order of arguments (self, target) is the reverse of os.link's.

**classmethod home**()

Return a new path pointing to the user's home directory (as returned by os.path.expanduser('~')).

**is_absolute**()

True if the path is absolute (has both a root and, if applicable, a drive).

**is_block_device**()

Whether this path is a block device.

**is_char_device**()

Whether this path is a character device.

**is_dir**()

Whether this path is a directory.

**is_fifo**()

Whether this path is a FIFO.

**is_file**()

Whether this path is a regular file (also True for symlinks pointing to regular files).

**is_junction**()

Whether this path is a junction.

**is_mount**()

Check if this path is a mount point

**is_relative_to**(*other*, */*, **_deprecated*)

Return True if the path is relative to another path or False.

**is_reserved**()

Return True if the path contains one of the special names reserved by the system, if any.

**is_socket**()

Whether this path is a socket.

**is_symlink**()

Whether this path is a symbolic link.

**iterdir**()

Yield path objects of the directory contents.

The children are yielded in arbitrary order, and the special entries '.' and '..' are not included.

**joinpath**(**pathsegments*)

Combine this path with one or several arguments, and return a new path representing either a subpath (if all arguments are relative paths) or a totally different path (if one of the arguments is anchored).

**joinuri**(*uri: str | os.PathLike[str]*) → *UPath*

Join with urljoin behavior for UPath instances

**lchmod**(*mode*)

Like chmod(), except if the path points to a symlink, the symlink's permissions are changed, rather than its target's.

**lstat**()

Like stat(), except if the path points to a symlink, the symlink's status information is returned, rather than its target's.

**match**(*path_pattern*, *, *case_sensitive=None*)

Return True if this path matches the given pattern.

**mkdir**(*mode=511*, *parents=False*, *exist_ok=False*)

Create a new directory at this given path.

**property name**

The final path component, if any.

**open**(*mode: str = 'r'*, **args: Any*, ***fsspec_kwargs: Any*) → IO[Any]

Open the file pointed by this path and return a file object, as the built-in open() function does.

**Parameters**

- **mode** – Opening mode. Default is 'r'.
- **buffering** – Default is the block size of the underlying fsspec filesystem.
- **encoding** – Encoding is only used in text mode. Default is None.
- **errors** – Error handling for encoding. Only used in text mode. Default is None.
- **newline** – Newline handling. Only used in text mode. Default is None.

- ****fsspec_kwargs** – Additional options for the fsspec filesystem.

**owner**()

Return the login name of the file owner.

**property parent**

The logical parent of the path.

**property parents**

A sequence of this path's logical parents.

**property parts**

An object providing sequence-like access to the components in the filesystem path.

**property path: str**

The path that a fsspec filesystem can use.

**property protocol: str**

The fsspec protocol for the path.

**read_bytes**()

Open the file in bytes mode, read it, and close the file.

**read_text**(*encoding=None*, *errors=None*)

Open the file in text mode, read it, and close the file.

**readlink**()

Return the path to which the symbolic link points.

**relative_to**(*other*, */*, **_deprecated*, *walk_up=False*)

Return the relative path to another path identified by the passed arguments. If the operation is not possible (because this is not related to the other path), raise ValueError.

The *walk_up* parameter controls whether .. may be used to resolve the path.

**rename**(*target*, ***, *recursive=False*, *maxdepth=None*, ***kwargs*)

Rename this path to the target path.

The target path may be absolute or relative. Relative paths are interpreted relative to the current working directory, *not* the directory of the Path object.

Returns the new Path instance pointing to the target path.

**replace**(*target*)

Rename this path to the target path, overwriting if that path exists.

The target path may be absolute or relative. Relative paths are interpreted relative to the current working directory, *not* the directory of the Path object.

Returns the new Path instance pointing to the target path.

**resolve**(*strict: bool = False*)

Make the path absolute, resolving all symlinks on the way and also normalizing it.

**rglob**(*pattern: str*, ***, *case_sensitive=None*)

Recursively yield all existing files (of any kind, including directories) matching the given relative pattern, anywhere in this subtree.

**rmdir**(*recursive: bool = True*)

Remove this directory. The directory must be empty.

**property root**

The root of the path, if any.

**samefile**(*other_path*)

Return whether other_path is the same or not as this file (as returned by os.path.samefile()).

**stat**(*, *follow_symlinks=True*) → *UPathStatResult*

Return the result of the stat() system call on this path, like os.stat() does.

**property stem**

The final path component, minus its last suffix.

**property storage_options: Mapping[str, Any]**

The fsspec storage options for the path.

**property suffix**

The final component's last suffix, if any.

This includes the leading period. For example: '.txt'

**property suffixes**

A list of the final component's suffixes, if any.

These include the leading periods. For example: ['.tar', '.gz']

**symlink_to**(*target*, *target_is_directory=False*)

Make this path a symlink pointing to the target path. Note the order of arguments (link, target) is the reverse of os.symlink.

**touch**(*mode=438*, *exist_ok=True*)

Create this file with the given access mode, if it doesn't exist.

**unlink**(*missing_ok=False*)

Remove this file or link. If the path is a directory, use rmdir() instead.

**walk**(*top_down=True*, *on_error=None*, *follow_symlinks=False*)

Walk the directory tree from this directory, similar to os.walk().

**with_name**(*name*)

Return a new path with the file name changed.

**with_segments**(**pathsegments*)

Construct a new path object from any number of path-like objects. Subclasses may override this method to customize how new path objects are created from methods like *iterdir()*.

**with_stem**(*stem*)

Return a new path with the stem changed.

**with_suffix**(*suffix*)

Return a new path with the file suffix changed. If the path has no suffix, add given suffix. If the given suffix is an empty string, remove the suffix from the path.

**write_bytes**(*data*)

Open the file in bytes mode, write to it, and close the file.

**write_text**(*data*, *encoding=None*, *errors=None*, *newline=None*)

Open the file in text mode, write to it, and close the file.

euporie.core.path.**parse_path**(*path: str | PathLike*, *resolve: bool = True*) → *Path*

Parse and resolve a path.

### euporie.core.processors

Buffer processors.

### Functions

| | |
|---|---|
| *cast*(typ, val) | Cast a value to a type. |
| *explode_text_fragments*(fragments) | Turn a list of (style_str, text) tuples into another list where each string is exactly one character. |
| *get_cwidth*(string) | Return width of a string. |

### euporie.core.processors.cast

euporie.core.processors.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.processors.explode_text_fragments

euporie.core.processors.**explode_text_fragments**(*fragments: Iterable[_T]*) → _ExplodedList[_T]

Turn a list of (style_str, text) tuples into another list where each string is exactly one character.

It should be fine to call this function several times. Calling this on a list that is already exploded, is a null operation.

**Parameters**

**fragments** – List of (style, text) tuples.

### euporie.core.processors.get_cwidth

euporie.core.processors.**get_cwidth**(*string: str*) → int

Return width of a string. Wrapper around `wcwidth`.

### Classes

| | |
|---|---|
| *AppendAutoSuggestion*([style]) | Append the auto suggestion to the input. |
| *AppendLineAutoSuggestion*([style]) | Append the auto suggestion to the current line of the input. |
| *CursorProcessor*(get_cursor_position[, char, ...]) | Show a mouse cursor. |
| *DiagnosticProcessor*(report[, style]) | Highlight diagnostics. |
| *Processor*() | Manipulate the fragments for a given line in a *BufferControl*. |
| *ShowTrailingWhiteSpaceProcessor*([char, style]) | Make trailing whitespace visible. |
| *Transformation*(fragments[, ...]) | Transformation result, as returned by `Processor.apply_transformation()`. |

### euporie.core.processors.AppendAutoSuggestion

**class** euporie.core.processors.**AppendAutoSuggestion**(*style: str = 'class:auto-suggestion'*)

Append the auto suggestion to the input. (The user can then press the right arrow the insert the suggestion.)

### euporie.core.processors.AppendLineAutoSuggestion

**class** euporie.core.processors.**AppendLineAutoSuggestion**(*style: str = 'class:auto-suggestion'*)

Append the auto suggestion to the current line of the input.

### euporie.core.processors.CursorProcessor

**class** euporie.core.processors.**CursorProcessor**(*get_cursor_position: Callable[[], Point], char: str = '', style: str = 'class:mouse'*)

Show a mouse cursor.

### euporie.core.processors.DiagnosticProcessor

**class** euporie.core.processors.**DiagnosticProcessor**(*report: Report | Callable[[], Report], style: str = 'underline'*)

Highlight diagnostics.

### euporie.core.processors.Processor

**class** euporie.core.processors.**Processor**

Manipulate the fragments for a given line in a `BufferControl`.

### euporie.core.processors.ShowTrailingWhiteSpaceProcessor

**class** euporie.core.processors.**ShowTrailingWhiteSpaceProcessor**(*char: str = '·', style: str = 'class:trailing-whitespace'*)

Make trailing whitespace visible.

### euporie.core.processors.Transformation

**class** euporie.core.processors.**Transformation**(*fragments: StyleAndTextTuples, source_to_display: SourceToDisplay | None = None, display_to_source: DisplayToSource | None = None*)

Transformation result, as returned by `Processor.apply_transformation()`.

**Important: Always make sure that the length of *document.text* is equal to** the length of all the text in *fragments*!

**Parameters**

- **fragments** – The transformed fragments. To be displayed, or to pass to the next processor.
- **source_to_display** – Cursor position transformation from original string to transformed string.
- **display_to_source** – Cursor position transformed from source string to original string.

**class** euporie.core.processors.**AppendLineAutoSuggestion**(*style: str = 'class:auto-suggestion'*)

Bases: *AppendAutoSuggestion*

Append the auto suggestion to the current line of the input.

**apply_transformation**(*ti: TransformationInput*) → *Transformation*

Inert fragments at the end of the current line.

**class** euporie.core.processors.**CursorProcessor**(*get_cursor_position: Callable[[], Point]*, *char: str = '?'*, *style: str = 'class:mouse'*)

Bases: *Processor*

Show a mouse cursor.

**apply_transformation**(*ti: TransformationInput*) → *Transformation*

Replace character at the cursor position.

**class** euporie.core.processors.**DiagnosticProcessor**(*report: Report | Callable[[], Report]*, *style: str = 'underline'*)

Bases: *Processor*

Highlight diagnostics.

**apply_transformation**(*ti: TransformationInput*) → *Transformation*

Underline the text ranges relating to diagnostics in the report.

**property report: *Report***

Return the current diagnostics report.

**class** euporie.core.processors.**ShowTrailingWhiteSpaceProcessor**(*char: str = '·'*, *style: str = 'class:trailing-whitespace'*)

Bases: *Processor*

Make trailing whitespace visible.

**apply_transformation**(*ti: TransformationInput*) → *Transformation*

Walk backwards through all the fragments and replace whitespace.

### euporie.core.pygments

Contain lexers for pygments.

### Classes

| | |
|---|---|
| *ArgparseLexer*(*args, **kwds) | A pygments lexer for agrparse help text. |
| *EuporiePygmentsStyle*() | Version of pygment's "native" style which works better on light backgrounds. |
| *RegexLexer*(*args, **kwds) | Base for simple stateful regular expression-based lexers. |
| *Style*() | |

#### euporie.core.pygments.ArgparseLexer

**class** euporie.core.pygments.**ArgparseLexer**(**args*, ***kwds*)

A pygments lexer for agrparse help text.

#### euporie.core.pygments.EuporiePygmentsStyle

**class** euporie.core.pygments.**EuporiePygmentsStyle**

Version of pygment's "native" style which works better on light backgrounds.

#### euporie.core.pygments.RegexLexer

**class** euporie.core.pygments.**RegexLexer**(**args*, ***kwds*)

Base for simple stateful regular expression-based lexers. Simplifies the lexing process so that you need only provide a list of states and regular expressions.

#### euporie.core.pygments.Style

**class** euporie.core.pygments.**Style**

**class** euporie.core.pygments.**ArgparseLexer**(**args*, ***kwds*)

Bases: *RegexLexer*

A pygments lexer for agrparse help text.

**add_filter**(*filter_*, ***options*)

Add a new stream filter to this lexer.

**alias_filenames = []**

A list of *fnmatch* patterns that match filenames which may or may not contain content for this lexer. This list is used by the *guess_lexer_for_filename()* function, to determine which lexers are then included in guessing the correct one. That means that e.g. every lexer for HTML and a template language should include `\*.html` in this list.

**aliases: ClassVar[list[str]] = ['argparse']**

A list of short, unique identifiers that can be used to look up the lexer from a list, e.g., using *get_lexer_by_name()*.

**static analyse_text**(*text*)

A static method which is called for lexer guessing.

It should analyse the text and return a float in the range from `0.0` to `1.0`. If it returns `0.0`, the lexer will not be selected as the most probable one, if it returns `1.0`, it will be selected immediately. This is used by *guess_lexer*.

The *LexerMeta* metaclass automatically wraps this function so that it works like a static method (no `self` or `cls` parameter) and the return value is automatically converted to *float*. If the return value is an object that is boolean *False* it's the same as if the return values was `0.0`.

**filenames: ClassVar[list[str]] = []**

A list of *fnmatch* patterns that match filenames which contain content for this lexer. The patterns in this list should be unique among all lexers.

**flags = 8**

Flags for compiling the regular expressions. Defaults to MULTILINE.

**get_tokens**(*text*, *unfiltered=False*)

This method is the basic interface of a lexer. It is called by the *highlight()* function. It must process the text and return an iterable of `(tokentype, value)` pairs from *text*.

Normally, you don't need to override this method. The default implementation processes the options recognized by all lexers (*stripnl*, *stripall* and so on), and then yields all tokens from *get_tokens_unprocessed()*, with the `index` dropped.

If *unfiltered* is set to *True*, the filtering mechanism is bypassed even if filters are defined.

**get_tokens_unprocessed**(*text*, *stack=('root',)*)

Split `text` into (tokentype, text) pairs.

`stack` is the initial stack (default: `['root']`)

**mimetypes = []**

A list of MIME types for content that can be lexed with this lexer.

**name = 'argparse'**

Full name of the lexer, in human-readable form

**priority = 0**

Priority, should multiple lexers match and no content is provided

**tokens: ClassVar[dict[str, list[tuple[str, pygments.token._TokenType] | tuple[str, pygments.token._TokenType, str]]]] = {'options': [('\\d+', Token.Literal.Number), (',', Token.Text), ('[^\\}]', Token.Literal.String), ('\\}', Token.Operator, '#pop')], 'root': [('(?<=usage: )[^\\s]+', Token.Name.Namespace), ('\\{', Token.Operator, 'options'), ('[\\[\\{\\|\\}\\]]', Token.Operator), ('((?<=\\s)|(?<=\\[))(--[a-zA-Z0-9-]+|-[a-zA-Z0-9-])', Token.Keyword), ('^(\\w+\\s)?\\w+:', Token.Generic.Heading), ('\\b(str|int|bool|UPath|loads)\\b', Token.Name.Builtin), ('\\b[A-Z]+_[A-Z]*\\b', Token.Name.Variable), ("'.*?'", Token.Literal.String), ('.', Token.Text)]}**

At all time there is a stack of states. Initially, the stack contains a single state 'root'. The top of the stack is called "the current state".

Dict of `{'state': [(regex, tokentype, new_state), ...], ...}`

`new_state` can be omitted to signify no state transition. If `new_state` is a string, it is pushed on the stack. This ensure the new current state is `new_state`. If `new_state` is a tuple of strings, all of those

strings are pushed on the stack and the current state will be the last element of the list. `new_state` can also be `combined('state1', 'state2', ...)` to signify a new, anonymous state combined from the rules of two or more existing ones. Furthermore, it can be '#pop' to signify going back one step in the state stack, or '#push' to push the current state on the stack again. Note that if you push while in a combined state, the combined state itself is pushed, and not only the state in which the rule is defined.

The tuple can also be replaced with `include('state')`, in which case the rules from the state named by the string are included in the current one.

**url = None**

URL of the language specification/definition. Used in the Pygments documentation. Set to an empty string to disable.

**version_added = None**

Version of Pygments in which the lexer was added.

**class** euporie.core.pygments.**EuporiePygmentsStyle**

Bases: *Style*

Version of pygment's "native" style which works better on light backgrounds.

**aliases = []**

**background_color = '#ffffff'**

overall background color (`None` means transparent)

**highlight_color = '#ffffcc'**

highlight background color

**line_number_background_color = 'transparent'**

line number background color

**line_number_color = 'inherit'**

line number font color

**line_number_special_background_color = '#ffffc0'**

special line number background color

**line_number_special_color = '#000000'**

special line number font color

**name = 'unnamed'**

```
styles: ClassVar[dict[pygments.token._TokenType, str]] = {Token: '',
Token.Comment: 'italic #888888', Token.Comment.Hashbang: '',
Token.Comment.Multiline: '', Token.Comment.Preproc: 'noitalic bold
#cd2828', Token.Comment.PreprocFile: '', Token.Comment.Single: '',
Token.Comment.Special: 'noitalic bold #e50808 bg:#520000', Token.Error:
'bold bg:#a61717 #ffffff', Token.Escape: '', Token.Generic: '',
Token.Generic.Deleted: '#d22323', Token.Generic.Emph: 'italic',
Token.Generic.EmphStrong: '', Token.Generic.Error: '#d22323',
Token.Generic.Heading: 'bold', Token.Generic.Inserted: '#589819',
Token.Generic.Output: '', Token.Generic.Prompt: '', Token.Generic.Strong:
'bold', Token.Generic.Subheading: 'underline', Token.Generic.Traceback:
'#d22323', Token.Keyword: 'bold #6ebf26', Token.Keyword.Constant: 'nobold
#ff3d3d', Token.Keyword.Declaration: '', Token.Keyword.Namespace: '',
Token.Keyword.Pseudo: 'nobold', Token.Keyword.Reserved: '',
Token.Keyword.Type: '', Token.Literal: '', Token.Literal.Date: '#2fbccd',
Token.Literal.Number: '#51b2fd', Token.Literal.Number.Bin: '',
Token.Literal.Number.Float: '', Token.Literal.Number.Hex: '',
Token.Literal.Number.Integer: '', Token.Literal.Number.Integer.Long: '',
Token.Literal.Number.Oct: '', Token.Literal.String: '#ed9d13',
Token.Literal.String.Affix: '', Token.Literal.String.Backtick: '',
Token.Literal.String.Char: '', Token.Literal.String.Delimiter: '',
Token.Literal.String.Doc: '', Token.Literal.String.Double: '',
Token.Literal.String.Escape: '', Token.Literal.String.Heredoc: '',
Token.Literal.String.Interpol: '', Token.Literal.String.Other: '#ffa500',
Token.Literal.String.Regex: '', Token.Literal.String.Single: '',
Token.Literal.String.Symbol: '', Token.Name: '', Token.Name.Attribute:
'noinherit', Token.Name.Builtin: '#2fbccd', Token.Name.Builtin.Pseudo: '',
Token.Name.Class: 'underline #71adff', Token.Name.Constant: '#40ffff',
Token.Name.Decorator: '#ffa500', Token.Name.Entity: '',
Token.Name.Exception: 'noinherit bold', Token.Name.Function: '#71adff',
Token.Name.Function.Magic: '', Token.Name.Label: '', Token.Name.Namespace:
'underline #71adff', Token.Name.Other: '', Token.Name.Property: '',
Token.Name.Tag: 'bold #6ebf26', Token.Name.Variable: '#40ffff',
Token.Name.Variable.Class: '', Token.Name.Variable.Global: '',
Token.Name.Variable.Instance: '', Token.Name.Variable.Magic: '',
Token.Operator: '', Token.Operator.Word: 'bold #6ebf26', Token.Other: '',
Token.Punctuation: '', Token.Punctuation.Marker: '', Token.Text: '',
Token.Text.Whitespace: ''}
```

Style definitions for individual token types.

```
web_style_gallery_exclude = False
```

### euporie.core.reference

Contain data reference dictionaries for value lookups.

### euporie.core.renderer

Extended version of prompt_toolkit's renderer.

## Functions

| | |
|---|---|
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |

### euporie.core.renderer.to_filter

euporie.core.renderer.**to_filter**(*bool_or_filter: Union[*Filter*, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

## Classes

| | |
|---|---|
| *BoundedWritePosition*(xpos, ypos, width, height) | A write position which also hold bounding box information. |
| *MouseHandlers*() | Two dimensional raster of callbacks for mouse events. |
| *Point*(x, y) | |
| *PtkRenderer* | alias of Renderer |
| *Renderer*(style, output[, full_screen, ...]) | Renderer with modifications. |
| *Screen*([default_char, initial_width, ...]) | Screen class which uses :py:`BoundedWritePosition`s. |
| *Size*(rows, columns) | |
| *Vt100_Output*(stdout, get_size[, term, ...]) | A Vt100 output which enables SGR pixel mouse positioning. |

### euporie.core.renderer.BoundedWritePosition

**class** euporie.core.renderer.**BoundedWritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*, *bbox:* euporie.core.data_structures.DiInt | *None* = *None*)

A write position which also hold bounding box information.

### euporie.core.renderer.MouseHandlers

**class** euporie.core.renderer.**MouseHandlers**

Two dimensional raster of callbacks for mouse events.

### euporie.core.renderer.Point

**class** euporie.core.renderer.**Point**(*x*, *y*)

### euporie.core.renderer.PtkRenderer

euporie.core.renderer.**PtkRenderer**

alias of Renderer

### euporie.core.renderer.Renderer

**class** euporie.core.renderer.**Renderer**(*style: BaseStyle*, *output: Output*, *full_screen: bool = False*, *mouse_support: FilterOrBool = False*, *cpr_not_supported_callback: Callable[[], None] | None = None*, *extend_height: FilterOrBool = False*, *extend_width: FilterOrBool = False*)

Renderer with modifications.

### euporie.core.renderer.Screen

**class** euporie.core.renderer.**Screen**(*default_char: prompt_toolkit.layout.screen.Char | None = None*, *initial_width: int = 0*, *initial_height: int = 0*)

Screen class which uses :py:`BoundedWritePosition`s.

### euporie.core.renderer.Size

**class** euporie.core.renderer.**Size**(*rows*, *columns*)

### euporie.core.renderer.Vt100_Output

**class** euporie.core.renderer.**Vt100_Output**(*stdout: TextIO*, *get_size: Callable[[], Size]*, *term: str | None = None*, *default_color_depth: prompt_toolkit.output.color_depth.ColorDepth | None = None*, *enable_bell: bool = True*, *enable_cpr: bool = True*)

A Vt100 output which enables SGR pixel mouse positioning.

**class** euporie.core.renderer.**Renderer**(*style:* BaseStyle, *output: Output*, *full_screen: bool = False*, *mouse_support: FilterOrBool = False*, *cpr_not_supported_callback: Callable[[], None] | None = None*, *extend_height: FilterOrBool = False*, *extend_width: FilterOrBool = False*)

Bases: Renderer

Renderer with modifications.

**CPR_TIMEOUT = 2**

**clear**() → None

Clear screen and go to 0,0

**erase**(*leave_alternate_screen: bool = True*) → None

Hide all output and put the cursor back at the first line. This is for instance used for running a system command (while hiding the CLI) and later resuming the same CLI.)

**Parameters**

**leave_alternate_screen** – When True, and when inside an alternate screen buffer, quit the alternate screen.

**property height_is_known: bool**

True when the height from the cursor until the bottom of the terminal is known. (It's often nicer to draw bottom toolbars only if the height is known, in order to avoid flickering when the CPR response arrives.)

**property last_rendered_screen: *prompt_toolkit.layout.screen.Screen* | None**

The *Screen* class that was generated during the last rendering. This can be *None*.

**render**(*app:* Application*[Any]*, *layout:* Layout, *is_done: bool = False*) → None

Render the current interface to the output.

**report_absolute_cursor_row**(*row: int*) → None

To be called when we know the absolute cursor position. (As an answer of a "Cursor Position Request" response.)

**request_absolute_cursor_position**() → None

Get current cursor position.

We do this to calculate the minimum available height that we can consume for rendering the prompt. This is the available space below te cursor.

For vt100: Do CPR request. (answer will arrive later.) For win32: Do API call. (Answer comes immediately.)

**reset**(*_scroll: bool = False*, *leave_alternate_screen: bool = True*) → None

Reset the output.

**property rows_above_layout: int**

Return the number of rows visible in the terminal above the layout.

**async wait_for_cpr_responses**(*timeout: int = 1*) → None

Wait for a CPR response.

**property waiting_for_cpr: bool**

Waiting for CPR flag. True when we send the request, but didn't got a response.

### euporie.core.style

Style related functions.

### Functions

| | |
|---|---|
| *build_style*(cp[, have_term_colors]) | Create an application style based on the given color palette. |
| *default_ui_style*() | Create a default *Style* object. |
| *hls_to_rgb*(h, l, s) | |
| *rgb_to_hls*(r, g, b) | |

### euporie.core.style.build_style

euporie.core.style.**build_style**(*cp:* ColorPalette, *have_term_colors:* *bool* = *True*) → *Style*

Create an application style based on the given color palette.

### euporie.core.style.default_ui_style

euporie.core.style.**default_ui_style**() → *BaseStyle*

Create a default *Style* object.

### euporie.core.style.hls_to_rgb

euporie.core.style.**hls_to_rgb**(*h*, *l*, *s*)

### euporie.core.style.rgb_to_hls

euporie.core.style.**rgb_to_hls**(*r*, *g*, *b*)

### Classes

| | |
|---|---|
| *ColorPalette*() | Define a collection of colors. |
| *ColorPaletteColor*(base[, _base_override]) | A representation of a color with adjustment methods. |
| *SimpleCache*([maxsize]) | Very simple cache that discards the oldest item when the cache size is exceeded. |
| *Style*(style_rules) | Create a `Style` instance from a list of style rules. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.style.ColorPalette

**class** euporie.core.style.**ColorPalette**

Define a collection of colors.

### euporie.core.style.ColorPaletteColor

**class** euporie.core.style.**ColorPaletteColor**(*base: str*, *_base_override: str = ''*)

A representation of a color with adjustment methods.

### euporie.core.style.SimpleCache

**class** euporie.core.style.**SimpleCache**(*maxsize: int = 8*)

Very simple cache that discards the oldest item when the cache size is exceeded.

**Parameters**

**maxsize** – Maximum size of the cache. (Don't make it too big.)

### euporie.core.style.Style

**class** euporie.core.style.**Style**(*style_rules: list[tuple[str, str]]*)

Create a `Style` instance from a list of style rules.

The *style_rules* is supposed to be a list of ('classnames', 'style') tuples. The classnames are a whitespace separated string of class names and the style string is just like a Pygments style definition, but with a few additions: it supports 'reverse' and 'blink'.

Later rules always override previous rules.

Usage:

```
Style([
    ('title', '#ff0000 bold underline'),
    ('something-else', 'reverse'),
    ('class1 class2', 'reverse'),
])
```

The `from_dict` classmethod is similar, but takes a dictionary as input.

### euporie.core.style.partial

**class** euporie.core.style.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.style.**ColorPalette**

Bases: `object`

Define a collection of colors.

**add_color**(*name: str*, *base: str*, *_base_override: str = ''*) → *ColorPalette*

Add a color to the palette.

**class** euporie.core.style.**ColorPaletteColor**(*base: str*, *_base_override: str = ''*)

Bases: object

A representation of a color with adjustment methods.

**adjust**(*hue: float = 0.0*, *brightness: float = 0.0*, *saturation: float = 0.0*, *rel: bool = True*) → *ColorPaletteColor*

Adjust the hue, saturation, or brightness of the color.

**Parameters**

- **hue** – The hue adjustment.
- **brightness** – The brightness adjustment.
- **saturation** – The saturation adjustment.
- **rel** – If True, perform a relative adjustment.

**Returns**
The adjusted color.

**Return type**
*ColorPaletteColor*

**darker**(*amount: float*, *rel: bool = True*) → *ColorPaletteColor*

Make the color darker.

**Parameters**

- **amount** – The amount to darken the color by.
- **rel** – If True, perform a relative adjustment.

**Returns**
The darker color.

**Return type**
*ColorPaletteColor*

**less**(*amount: float*, *rel: bool = True*) → *ColorPaletteColor*

Make bright colors brighter and dark colors darker.

**Parameters**

- **amount** – The amount to adjust the color by.
- **rel** – If True, perform a relative adjustment.

**Returns**
The adjusted color.

**Return type**
*ColorPaletteColor*

**lighter**(*amount: float*, *rel: bool = True*) → *ColorPaletteColor*

Make the color lighter.

**Parameters**

- **amount** – The amount to lighten the color by.
- **rel** – If True, perform a relative adjustment.

**Returns**
The lighter color.

**Return type**
*ColorPaletteColor*

**more**(*amount: float*, *rel: bool = True*) → *ColorPaletteColor*

Make bright colors darker and dark colors brighter.

**Parameters**

- **amount** – The amount to adjust the color by.
- **rel** – If True, perform a relative adjustment.

**Returns**
The adjusted color.

**Return type**
*ColorPaletteColor*

**towards**(*other:* ColorPaletteColor, *amount: float*) → *ColorPaletteColor*

Interpolate between two colors.

euporie.core.style.**build_style**(*cp:* ColorPalette, *have_term_colors: bool = True*) → *Style*

Create an application style based on the given color palette.

### euporie.core.suggest

Suggest line completions from kernel history.

### Functions

| | |
|---|---|
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |

### euporie.core.suggest.to_filter

euporie.core.suggest.**to_filter**(*bool_or_filter: Union[*Filter, *bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### Classes

| | |
|---|---|
| *AutoSuggest*() | Base class for auto suggestion implementations. |
| *ConditionalAutoSuggest*(auto_suggest, filter) | Auto suggest that can be turned on and of according to a certain condition. |
| *ConditionalAutoSuggestAsync*(auto_suggest, filter) | Auto suggest that can be turned on and of according to a certain condition. |
| *HistoryAutoSuggest*(history[, cache_size]) | Suggest line completions from a `History` object. |
| *KernelAutoSuggest*(kernel) | Suggest line completions from kernel history. |
| *Suggestion*(text) | Suggestion returned by an auto-suggest algorithm. |
| *deque* | deque([iterable[, maxlen]]) --> deque object |

### euporie.core.suggest.AutoSuggest

**class** euporie.core.suggest.**AutoSuggest**

Base class for auto suggestion implementations.

### euporie.core.suggest.ConditionalAutoSuggest

**class** euporie.core.suggest.**ConditionalAutoSuggest**(*auto_suggest:* AutoSuggest, *filter: bool* | prompt_toolkit.filters.base.Filter)

Auto suggest that can be turned on and of according to a certain condition.

### euporie.core.suggest.ConditionalAutoSuggestAsync

**class** euporie.core.suggest.**ConditionalAutoSuggestAsync**(*auto_suggest:* AutoSuggest, *filter: bool* | Filter)

Auto suggest that can be turned on and of according to a certain condition.

### euporie.core.suggest.HistoryAutoSuggest

**class** euporie.core.suggest.**HistoryAutoSuggest**(*history:* History, *cache_size: int = 100000*)

Suggest line completions from a `History` object.

### euporie.core.suggest.KernelAutoSuggest

**class** euporie.core.suggest.**KernelAutoSuggest**(*kernel:* Kernel)

Suggest line completions from kernel history.

### euporie.core.suggest.Suggestion

**class** euporie.core.suggest.**Suggestion**(*text: str*)

Suggestion returned by an auto-suggest algorithm.

**Parameters**

**text** – The suggestion text.

### euporie.core.suggest.deque

**class** euporie.core.suggest.**deque**

deque([iterable[, maxlen]]) –> deque object

A list-like sequence optimized for data accesses near its endpoints.

**class** euporie.core.suggest.**ConditionalAutoSuggestAsync**(*auto_suggest:* AutoSuggest, *filter: bool* | Filter)

Bases: *ConditionalAutoSuggest*

Auto suggest that can be turned on and of according to a certain condition.

**get_suggestion**(*buffer:* Buffer, *document:* Document) → *Suggestion* | None

Return *None* or a *Suggestion* instance.

We receive both *Buffer* and *Document*. The reason is that auto suggestions are retrieved asynchronously. (Like completions.) The buffer text could be changed in the meantime, but `document` contains the buffer document like it was at the start of the auto suggestion call. So, from here, don't access `buffer.text`, but use `document.text` instead.

**Parameters**

- **buffer** – The *Buffer* instance.
- **document** – The *Document* instance.

**async get_suggestion_async**(*buffer:* Buffer, *document:* Document) → *Suggestion* | None

Get suggestions asynchronously if the filter allows.

**class** euporie.core.suggest.**HistoryAutoSuggest**(*history:* History, *cache_size:* *int* = *100000*)

Bases: *AutoSuggest*

Suggest line completions from a `History` object.

**get_suggestion**(*buffer:* Buffer, *document:* Document) → *Suggestion* | None

Get a line completion suggestion.

**async get_suggestion_async**(*buff:* Buffer, *document:* Document) → *Suggestion* | None

Return a `Future` which is set when the suggestions are ready. This function can be overloaded in order to provide an asynchronous implementation.

**lookup_suggestion**(*line:* *str*) → *prompt_toolkit.auto_suggest.Suggestion* | None

Find the most recent matching line in the history.

**class** euporie.core.suggest.**KernelAutoSuggest**(*kernel:* Kernel)

Bases: *AutoSuggest*

Suggest line completions from kernel history.

**get_suggestion**(*buffer:* Buffer, *document:* Document) → *Suggestion* | None

Doe nothing.

**async get_suggestion_async**(*buff:* Buffer, *document:* Document) → *Suggestion* | None

Return suggestions based on matching kernel history.

## euporie.core.tabs

Contain various application tab implementations.

### Modules

| | |
|---|---|
| *euporie.core.tabs.base* | Contain tab base classes. |
| *euporie.core.tabs.notebook* | Contain the main class for a notebook file. |

### euporie.core.tabs.base

Contain tab base classes.

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *get_app*() | Get the current active (running) Application. |
| *open_comm*(comm_container, content, buffers) | Create a new object respsenting a Comm. |
| *parse_path*(path[, resolve]) | Parse and resolve a path. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *run_in_thread_with_context*(func, *args[, daemon]) | Run a function in an thread, but make sure it uses the same contextvars. |

### euporie.core.tabs.base.add_cmd

euporie.core.tabs.base.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.tabs.base.add_setting

euporie.core.tabs.base.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.tabs.base.get_app

euporie.core.tabs.base.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.tabs.base.open_comm

euporie.core.tabs.base.**open_comm**(*comm_container: KernelTab*, *content: dict[str, Any]*, *buffers: Sequence[bytes]*) → *Comm*

Create a new object respsenting a Comm.

The class used to represent the Comm is determined by the "target_class" given in the `comm_open` message.

**Parameters**

- **comm_container** – The notebook this comm belongs to
- **content** – The content of the `comm_open` message

- **buffers** – A list of binary data buffers sent with the `comm_open` message

**Returns**
A class representing the comm

### euporie.core.tabs.base.parse_path

`euporie.core.tabs.base.`**`parse_path`**(*path: str | PathLike*, *resolve: bool = True*) → *Path*

Parse and resolve a path.

### euporie.core.tabs.base.register_bindings

`euporie.core.tabs.base.`**`register_bindings`**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.tabs.base.run_in_thread_with_context

`euporie.core.tabs.base.`**`run_in_thread_with_context`**(*func: Callable*, **args: Any*, *daemon: bool = True*, ***kwargs: Any*) → None

Run a function in an thread, but make sure it uses the same contextvars.

This is required so that the function will see the right application.

## Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *DeduplicateCompleter*(completer) | Asynchronous wrapper around a completer that removes duplicates. |
| *DummyAutoSuggest*() | AutoSuggest class that doesn't return any suggestion. |
| *DummyHistory*() | *History* object that doesn't remember anything. |
| *DynamicCompleter*(get_completer) | Completer class that can dynamically returns any Completer. |
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *FirstInspector*(inspectors) | Return results of the first inspector to response. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *HistoryAutoSuggest*(history[, cache_size]) | Suggest line completions from a `History` object. |
| *InMemoryHistory*([history_strings]) | *History* class that keeps a list of all strings in memory. |
| *Kernel*(kernel_tab[, threaded, allow_stdin, ...]) | Run a notebook kernel and communicates with it asynchronously. |
| *KernelCompleter*(kernel) | A prompt_toolkit completer which provides completions from a Jupyter kernel. |
| *KernelHistory*(kernel[, n]) | Load the kernel's command history. |
| *KernelInput*(kernel_tab[, text, multiline, ...]) | Kernel input text areas. |
| *KernelInspector*(kernel) | Inspector which retrieves contextual help from a Jupyter kernel. |
| *KernelTab*(app[, path, kernel, comms, ...]) | A Tab which connects to a kernel. |
| *LspCompleter*(lsp, path) | A completer for documents using an LSP. |
| *LspFormatter*(lsp, path[, languages]) | Format a document using a LSP server. |
| *LspInspector*(lsp, path) | Inspector which retrieves contextual help from a Language Server. |
| *MsgCallbacks* | Typed dictionary for named message callbacks. |
| *Report*([iterable]) | Class for storing a diagnostic report. |
| *Tab*(app[, path]) | Base class for interface tabs. |
| *UPath*(*args[, protocol]) | |
| *WeakKeyDictionary*([dict]) | Mapping class that references keys weakly. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *WindowAlign*(value[, names, module, ...]) | Alignment of the Window content. |
| *deque* | deque([iterable[, maxlen]]) --> deque object |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.tabs.base.ABCMeta

**class** euporie.core.tabs.base.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.tabs.base.DeduplicateCompleter

**class** euporie.core.tabs.base.**DeduplicateCompleter**(*completer:* Completer)

Asynchronous wrapper around a completer that removes duplicates.

Only the first unique completions are kept. Completions are considered to be a duplicate if they result in the same document text when they would be applied.

### euporie.core.tabs.base.DummyAutoSuggest

**class** euporie.core.tabs.base.**DummyAutoSuggest**

AutoSuggest class that doesn't return any suggestion.

### euporie.core.tabs.base.DummyHistory

**class** euporie.core.tabs.base.**DummyHistory**

*History* object that doesn't remember anything.

### euporie.core.tabs.base.DynamicCompleter

**class** euporie.core.tabs.base.**DynamicCompleter**(*get_completer: Callable[[],* prompt_toolkit.completion.base.Completer | *None]*)

Completer class that can dynamically returns any Completer.

**Parameters**

**get_completer** – Callable that returns a *Completer* instance.

### euporie.core.tabs.base.Event

**class** euporie.core.tabs.base.**Event**(*sender: _Sender*, *handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.tabs.base.FirstInspector

**class** euporie.core.tabs.base.**FirstInspector**(*inspectors: Sequence[*Inspector*] | Callable[[], Sequence[*Inspector*]]*)

Return results of the first inspector to response.

### euporie.core.tabs.base.FormattedTextControl

**class** euporie.core.tabs.base.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[],* Point | *None]* | *None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **key_bindings** – a *KeyBindings* object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.tabs.base.HistoryAutoSuggest

**class** euporie.core.tabs.base.**HistoryAutoSuggest**(*history:* History, *cache_size:* *int* = *100000*)

Suggest line completions from a `History` object.

### euporie.core.tabs.base.InMemoryHistory

**class** euporie.core.tabs.base.**InMemoryHistory**(*history_strings:* *Optional[Sequence[str]]* = *None*)

*`History`* class that keeps a list of all strings in memory.

In order to prepopulate the history, it's possible to call either *append_string* for all items or pass a list of strings to *__init__* here.

### euporie.core.tabs.base.Kernel

**class** euporie.core.tabs.base.**Kernel**(*kernel_tab:* KernelTab, *threaded:* *bool* = *True*, *allow_stdin:* *bool* = *False*, *default_callbacks:* MsgCallbacks | *None* = *None*, *connection_file:* Path | *None* = *None*)

Run a notebook kernel and communicates with it asynchronously.

Has the ability to run itself in it's own thread.

### euporie.core.tabs.base.KernelCompleter

**class** euporie.core.tabs.base.**KernelCompleter**(*kernel:* Kernel)

A prompt_toolkit completer which provides completions from a Jupyter kernel.

### euporie.core.tabs.base.KernelHistory

**class** euporie.core.tabs.base.**KernelHistory**(*kernel:* Kernel, *n:* *int* = *1000*)

Load the kernel's command history.

### euporie.core.tabs.base.KernelInput

**class** euporie.core.tabs.base.**KernelInput**(*kernel_tab: KernelTab*, *text: str = ''*, *multiline: FilterOrBool = True*, *password: FilterOrBool = False*, *lexer: Lexer | None = None*, *auto_suggest: AutoSuggest | None = None*, *completer: Completer | None = None*, *complete_while_typing: FilterOrBool = True*, *validator: Validator | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *history: History | None = None*, *focusable: FilterOrBool = True*, *focus_on_click: FilterOrBool = True*, *wrap_lines: FilterOrBool = False*, *read_only: FilterOrBool = False*, *width: AnyDimension = None*, *height: AnyDimension = None*, *dont_extend_height: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *line_numbers: bool = False*, *get_line_prefix: GetLinePrefixCallable | None = None*, *scrollbar: FilterOrBool = True*, *style: str = 'class:kernel-input'*, *search_field: SearchToolbar | None = None*, *preview_search: FilterOrBool = False*, *prompt: AnyFormattedText = ''*, *input_processors: list[Processor] | None = None*, *name: str = ''*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *on_cursor_position_changed: Callable[[Buffer], None] | None = None*, *tempfile_suffix: str | Callable[[], str] = ''*, *key_bindings: KeyBindingsBase | None = None*, *enable_history_search: FilterOrBool = False*, *autosuggest_while_typing: FilterOrBool = True*, *validate_while_typing: FilterOrBool = False*, *scroll_offsets: ScrollOffsets | None = None*, *formatters: list[Formatter] | None = None*, *language: str | Callable[[], str] | None = None*, *diagnostics: Report | Callable[[], Report] | None = None*, *inspector: Inspector | None = None*, *show_diagnostics: FilterOrBool = True*)

Kernel input text areas.

A customized text area for the cell input.

### euporie.core.tabs.base.KernelInspector

**class** euporie.core.tabs.base.**KernelInspector**(*kernel: Kernel*)

Inspector which retrieves contextual help from a Jupyter kernel.

### euporie.core.tabs.base.KernelTab

**class** euporie.core.tabs.base.**KernelTab**(*app: BaseApp*, *path: Path | None = None*, *kernel: Kernel | None = None*, *comms: dict[str, Comm] | None = None*, *use_kernel_history: bool = False*, *connection_file: Path | None = None*)

A Tab which connects to a kernel.

### euporie.core.tabs.base.LspCompleter

**class** euporie.core.tabs.base.**LspCompleter**(*lsp:* LspClient, *path:* Path)

A completer for documents using an LSP.

### euporie.core.tabs.base.LspFormatter

**class** euporie.core.tabs.base.**LspFormatter**(*lsp:* LspClient, *path:* Path, *languages: set[str]* | *None* = *None*)

Format a document using a LSP server.

### euporie.core.tabs.base.LspInspector

**class** euporie.core.tabs.base.**LspInspector**(*lsp:* LspClient, *path:* Path)

Inspector which retrieves contextual help from a Language Server.

### euporie.core.tabs.base.MsgCallbacks

**class** euporie.core.tabs.base.**MsgCallbacks**

Typed dictionary for named message callbacks.

### euporie.core.tabs.base.Report

**class** euporie.core.tabs.base.**Report**(*iterable=()*, */*)

Class for storing a diagnostic report.

### euporie.core.tabs.base.Tab

**class** euporie.core.tabs.base.**Tab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Base class for interface tabs.

### euporie.core.tabs.base.UPath

**class** euporie.core.tabs.base.**UPath**(**args*, *protocol: str* | *None* = *None*, ***storage_options: Any*)

### euporie.core.tabs.base.WeakKeyDictionary

**class** euporie.core.tabs.base.**WeakKeyDictionary**(*dict=None*)

Mapping class that references keys weakly.

Entries in the dictionary will be discarded when there is no longer a strong reference to the key. This can be used to associate additional data with an object owned by other parts of an application without adding attributes to those objects. This can be especially useful with objects that override attribute accesses.

### euporie.core.tabs.base.Window

**class** euporie.core.tabs.base.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.core.tabs.base.WindowAlign

**class** euporie.core.tabs.base.**WindowAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of the Window content.

Note that this is different from *HorizontalAlign* and *VerticalAlign*, which are used for the alignment of the child containers in respectively *VSplit* and *HSplit*.

### euporie.core.tabs.base.deque

**class** euporie.core.tabs.base.**deque**

deque([iterable[, maxlen]]) –> deque object

A list-like sequence optimized for data accesses near its endpoints.

### euporie.core.tabs.base.partial

**class** euporie.core.tabs.base.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.tabs.base.**KernelTab**(*app: BaseApp*, *path: Path | None = None*, *kernel: Kernel | None = None*, *comms: dict[str, Comm] | None = None*, *use_kernel_history: bool = False*, *connection_file: Path | None = None*)

Bases: *Tab*

A Tab which connects to a kernel.

**allow_stdin: bool**

**bg_init = True**

**change_kernel**(*msg: str | None = None*, *startup: bool = False*) → None
Prompt the user to select a new kernel.

**close**(*cb: Callable | None = None*) → None
Shut down kernel when tab is closed.

**comm_close**(*content: dict*, *buffers: Sequence[bytes]*) → None
Close a notebook Comm.

**comm_msg**(*content: dict*, *buffers: Sequence[bytes]*) → None
Respond to a Comm message from the kernel.

**comm_open**(*content: dict*, *buffers: Sequence[bytes]*) → None
Register a new kernel Comm object in the notebook.

**container: AnyContainer**

**property current_input: *KernelInput***
Return the currently active kernel input, if any.

**default_callbacks: *MsgCallbacks***

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None
Focus the tab (or make it visible).

**init_kernel**(*kernel:* Kernel | *None = None*, *comms: dict[str,* Comm*] | None = None*, *use_kernel_history: bool = False*, *connection_file:* Path | *None = None*) → None
Set up the tab's kernel and related components.

**interrupt_kernel**() → None
Interrupt the current *Notebook*'s kernel.

**kernel: *Kernel***

**property kernel_display_name: str**
Return the display name of the kernel defined in the notebook JSON.

**property kernel_lang_file_ext: str**
Return the display name of the kernel defined in the notebook JSON.

**kernel_language: str**

**property kernel_name: str**
Return the name of the kernel defined in the notebook JSON.

**kernel_started**(*result: dict[str, Any] | None = None*) → None
Task to run when the kernel has started.

**property language: str**
Return the name of the kernel defined in the notebook JSON.

**async load_history**() → None
Load kernel history.

**async load_lsps**() → None
Load the LSP clients.

**lsp_after_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we saved a document.

**lsp_before_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we are about to save a document.

**lsp_change_handler**(*lsp:* LspClient) → None
Tell the LSP server a file has changed.

**lsp_close_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_open_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_update_diagnostics**(*lsp:* LspClient) → None
Process a new diagnostic report from the LSP.

**property metadata: dict[str, Any]**
Return a dictionary to hold notebook / kernel metadata.

**mime_types: ClassVar[set[str]] = {}**

**name: str | None = None**

**post_init_kernel**() → None
Run stuff after the kernel is loaded.

**pre_init_kernel**() → None
Run stuff before the kernel is loaded.

**report**() → *Report*
Return the current diagnostic reports.

**report_kernel_error**(*error: Exception | None*) → None
Report a kernel error to the user.

**reset**() → None
Reset the state of the tab.

**restart_kernel**(*cb: Callable | None = None*) → None
Restart the current *Notebook*'s kernel.

**save**(*path:* Path | *None = None*, *cb: Callable | None = None*) → None
Save the current notebook.

**set_kernel_info**(*info: dict*) → None
Handle kernel info requests.

**property title: str**
Return the tab title.

**weight: int = 0**

**class** euporie.core.tabs.base.**Tab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Bases: object

Base class for interface tabs.

**close**(*cb: Callable* | *None* = *None*) → None

Close a tab with a callback.

**Parameters**

**cb** – A function to call after the tab is closed.

**container: AnyContainer**

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None

Focus the tab (or make it visible).

**mime_types: ClassVar[set[str]] = {}**

**name: str | None = None**

**reset**() → None

Reset the state of the tab.

**save**(*path:* Path | *None* = *None*, *cb: Callable* | *None* = *None*) → None

Save the current notebook.

**property title: str**

Return the tab title.

**weight: int = 0**

### euporie.core.tabs.notebook

Contain the main class for a notebook file.

### Functions

| | |
|---|---|
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *edit_in_editor*(filename[, line_number]) | Suspend the current app and edit a file in an external editor. |
| *get_cell_id*(cell_json) | Return the cell ID field defined in a cell JSON object. |
| *get_cmd*(name) | Get a command from the centralized command system by name. |
| *open_comm*(comm_container, content, buffers) | Create a new object respsenting a Comm. |
| *read_nb*(fp[, as_version, fmt, config]) | Read a notebook from a file name or a file object |
| *standard_b64decode*(s) | Decode bytes encoded with the standard Base64 alphabet. |
| *write_nb*(nb, fp[, version, fmt, config]) | Write a notebook to a file name or a file object |

### euporie.core.tabs.notebook.abstractmethod

euporie.core.tabs.notebook.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):

…

### euporie.core.tabs.notebook.add_setting

euporie.core.tabs.notebook.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.tabs.notebook.edit_in_editor

euporie.core.tabs.notebook.**edit_in_editor**(*filename: str*, *line_number: int = 0*) → None

Suspend the current app and edit a file in an external editor.

### euporie.core.tabs.notebook.get_cell_id

euporie.core.tabs.notebook.**get_cell_id**(*cell_json: dict*) → str

Return 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

### euporie.core.tabs.notebook.get_cmd

euporie.core.tabs.notebook.**get_cmd**(*name: str*) → *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.core.tabs.notebook.open_comm

euporie.core.tabs.notebook.**open_comm**(*comm_container: KernelTab*, *content: dict[str, Any]*, *buffers: Sequence[bytes]*) → *Comm*

Create a new object respsenting a Comm.

The class used to represent the Comm is determined by the "target_class" given in the `comm_open` message.

**Parameters**

- **comm_container** – The notebook this comm belongs to
- **content** – The content of the `comm_open` message
- **buffers** – A list of binary data buffers sent with the `comm_open` message

**Returns**
A class representing the comm

### euporie.core.tabs.notebook.read_nb

euporie.core.tabs.notebook.**read_nb**(*fp*, *as_version=nbformat.NO_CONVERT*, *fmt=None*, *config=None*, ***kwargs*)

Read a notebook from a file name or a file object

**Parameters**

- **fp** – a file name or a file object
- **as_version** – see nbformat.read
- **fmt** – (optional) the jupytext format like *md*, *py:percent*, …
- **config** – (optional) a Jupytext configuration object
- **kwargs** – (not used) additional parameters for nbformat.read

**Returns**
the notebook

### euporie.core.tabs.notebook.standard_b64decode

euporie.core.tabs.notebook.**standard_b64decode**(*s*)

Decode bytes encoded with the standard Base64 alphabet.

Argument s is a bytes-like object or ASCII string to decode. The result is returned as a bytes object. A binascii.Error is raised if the input is incorrectly padded. Characters that are not in the standard alphabet are discarded prior to the padding check.

### euporie.core.tabs.notebook.write_nb

euporie.core.tabs.notebook.**write_nb**(*nb*, *fp*, *version=nbformat.NO_CONVERT*, *fmt=None*, *config=None*, ***kwargs*)

Write a notebook to a file name or a file object

**Parameters**

- **nb** – the notebook
- **fp** – a file name or a file object
- **version** – see nbformat.write
- **fmt** – (optional if fp is a file name) the jupytext format like *md*, *py:percent*, …
- **config** – (optional) a Jupytext configuration object
- **kwargs** – (not used) additional parameters for nbformat.write

## Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *BaseNotebook*(app[, path, kernel, comms, ...]) | The main notebook container class. |
| *Cell*(index, json, kernel_tab[, is_new]) | A kernel_tab cell element. |
| *KernelTab*(app[, path, kernel, comms, ...]) | A Tab which connects to a kernel. |
| *MsgCallbacks* | Typed dictionary for named message callbacks. |
| *Never*() | Never enable feature. |
| *UntitledPath*(*args[, protocol]) | A path for untitled files, as needed for LSP servers. |
| *abstractproperty*([fget, fset, fdel, doc]) | A decorator indicating abstract properties. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.tabs.notebook.ABCMeta

**class** euporie.core.tabs.notebook.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.tabs.notebook.BaseNotebook

**class** euporie.core.tabs.notebook.**BaseNotebook**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *False*, *json: dict[str, Any]* | *None* = *None*)

The main notebook container class.

### euporie.core.tabs.notebook.Cell

**class** euporie.core.tabs.notebook.**Cell**(*index: int*, *json: dict*, *kernel_tab:* BaseNotebook, *is_new: bool* = *False*)

A kernel_tab cell element.

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

### euporie.core.tabs.notebook.KernelTab

**class** euporie.core.tabs.notebook.**KernelTab**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *False*, *connection_file:* Path | *None* = *None*)

A Tab which connects to a kernel.

### euporie.core.tabs.notebook.MsgCallbacks

**class** euporie.core.tabs.notebook.**MsgCallbacks**

Typed dictionary for named message callbacks.

### euporie.core.tabs.notebook.Never

**class** euporie.core.tabs.notebook.**Never**

Never enable feature.

### euporie.core.tabs.notebook.UntitledPath

**class** euporie.core.tabs.notebook.**UntitledPath**(**args*, *protocol: str* | *None* = *None*, ***storage_options: Any*)

A path for untitled files, as needed for LSP servers.

### euporie.core.tabs.notebook.abstractproperty

**class** euporie.core.tabs.notebook.**abstractproperty**(*fget=None*, *fset=None*, *fdel=None*, *doc=None*)

A decorator indicating abstract properties.

Deprecated, use 'property' with 'abstractmethod' instead:

**class C(ABC):**
@property @abstractmethod def my_abstract_property(self):

…

### euporie.core.tabs.notebook.partial

**class** euporie.core.tabs.notebook.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.tabs.notebook.**BaseNotebook**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *False*, *json: dict[str, Any]* | *None* = *None*)

Bases: *KernelTab*

The main notebook container class.

**allow_stdin: bool = False**

**bg_init = True**

**abstract property cell: *Cell***

Return the current cell.

**change_kernel**(*msg: str* | *None* = *None*, *startup: bool* = *False*) → None

Prompt the user to select a new kernel.

**close**(*cb: Callable* | *None* = *None*) → None

Check if the user want to save an unsaved notebook, then close the file.

**Parameters**
**cb** – A callback to run if after closing the notebook.

**comm_close**(*content: dict*, *buffers: Sequence[bytes]*) → None

Close a notebook Comm.

**comm_msg**(*content: dict*, *buffers: Sequence[bytes]*) → None

Respond to a Comm message from the kernel.

**comm_open**(*content: dict*, *buffers: Sequence[bytes]*) → None

Register a new kernel Comm object in the notebook.

**comms: dict[str, *Comm*]**

**completers: list[*Completer*]**

**container: AnyContainer**

**property current_input: *KernelInput***

Return the currently active kernel input, if any.

**default_callbacks: *MsgCallbacks***

**edit_mode = False**

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None

Focus the tab (or make it visible).

**formatters: list[*Formatter*]**

**get_cell_by_id**(*cell_id: str*) → *euporie.core.widgets.cell.Cell* | None

Return a reference to the *Cell* container with a given cell id.

**history: *History***

**init_kernel**(*kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *False*, *connection_file:* Path | *None* = *None*) → None

Set up the tab's kernel and related components.

**inspectors: list[*Inspector*]**

**interrupt_kernel**() → None

Interrupt the current *Notebook*'s kernel.

**kernel: *Kernel***

**kernel_died**() → None

Call if the kernel dies.

**property kernel_display_name: str**

Return the display name of the kernel defined in the notebook JSON.

**property kernel_lang_file_ext: str**

Return the display name of the kernel defined in the notebook JSON.

**kernel_language: str**

**property kernel_name: str**

Return the name of the kernel defined in the notebook JSON.

**kernel_started**(*result: dict[str, Any]* | *None* = *None*) → None

Task to run when the kernel has started.

**lang_file_ext**() → str

Return the file extension for scripts in the notebook's language.

**property language: str**

Return the name of the kernel defined in the notebook JSON.

**load**() → None

Load the notebook file from the file-system.

**abstract load_container**() → AnyContainer

Abcract method for loading the notebook's main container.

**async load_history**() → None
Load kernel history.

**async load_lsps**() → None
Load the LSP clients.

**load_widgets_from_metadata**() → None
Load widgets from state saved in notebook metadata.

**lsp_after_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we saved a document.

**lsp_before_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we are about to save a document.

**lsp_change_handler**(*lsp:* LspClient) → None
Tell the LSP server a file metadata has changed.

**lsp_close_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_open_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_update_diagnostics**(*lsp:* LspClient) → None
Process a new diagnostic report from the LSP.

**lsps: list[*LspClient*]**

**property metadata: dict[str, Any]**
Return a dictionary to hold notebook / kernel metadata.

**mime_types: ClassVar[set[str]] = {}**

**name: str | None = None**

**property path_name: str**
Return the path name.

**post_init_kernel**() → None
Load the notebook container after the kernel has been loaded.

**pre_init_kernel**() → None
Run stuff before the kernel is loaded.

**refresh**(*slice_: slice | None = None*, *scroll: bool = False*) → None
Refresh the notebook.

**refresh_cell**(*cell:* Cell) → None
Trigger the refresh of a notebook cell.

**rendered_cells**() → list[*euporie.core.widgets.cell.Cell*]
Return a list of rendered notebooks' cells.

**report**() → *Report*
Return the current diagnostic reports.

**report_kernel_error**(*error: Exception | None*) → None
Report a kernel error to the user.

**reports**: *WeakKeyDictionary*[*LspClient*, *Report*]

**reset**() → None

Reload the notebook file from the disk and re-render.

**restart_kernel**(*cb: Callable | None = None*) → None

Restart the current *Notebook*'s kernel.

**run_cell**(*cell:* Cell, *wait: bool = False*, *callback: Callable[..., None] | None = None*) → None

Run a cell.

**Parameters**

- **cell** – The rendered cell to run. If `None`, runs the currently selected cell.
- **wait** – If `True`, blocks until cell execution is finished
- **callback** – Function to run after completion

**save**(*path:* Path *| None = None*, *cb: Callable | None = None*) → None

Write the notebook's JSON to the current notebook's file.

Additionally save the widget state to the notebook metadata.

**Parameters**

- **path** – An optional new path at which to save the tab
- **cb** – A callback to run if after saving the notebook.

**scroll_to**(*index: int*) → None

Scroll to a cell by index.

**select**(*cell_index: int*, *extend: bool = False*, *position: int | None = None*, *scroll: bool = False*) → None

Select a cell.

**property selected_indices**: **list**[**int**]

Return a list of the currently selected cell indices.

**set_kernel_info**(*info: dict*) → None

Handle kernel info requests.

**set_status**(*status: str*) → None

Call when kernel status changes.

**suggester**: *AutoSuggest*

**property title**: **str**

Return the tab title.

**weight**: **int** = **0**

### euporie.core.terminal

Contain classes related to querying terminal features.

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *b64decode*(s[, altchars, validate]) | Decode the Base64 encoded bytes-like object or ASCII string s. |
| *edit_in_editor*(filename[, line_number]) | Suspend the current app and edit a file in an external editor. |
| *extend_enum*(enumeration, name, *args, **kwds) | Add a new member to an existing Enum. |
| *get_app*() | Get the current active (running) Application. |
| *lru_cache*([maxsize, typed]) | Least-recently-used cache decorator. |
| *passthrough*(cmd) | Wrap an escape sequence for terminal passthrough. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *run_in_terminal*(func[, render_cli_done, ...]) | Run function on the terminal above the current application or prompt. |

### euporie.core.terminal.add_cmd

euporie.core.terminal.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.terminal.b64decode

euporie.core.terminal.**b64decode**(*s*, *altchars=None*, *validate=False*)

Decode the Base64 encoded bytes-like object or ASCII string s.

Optional altchars must be a bytes-like object or ASCII string of length 2 which specifies the alternative alphabet used instead of the '+' and '/' characters.

The result is returned as a bytes object. A binascii.Error is raised if s is incorrectly padded.

If validate is False (the default), characters that are neither in the normal base-64 alphabet nor the alternative alphabet are discarded prior to the padding check. If validate is True, these non-alphabet characters in the input result in a binascii.Error. For more information about the strict base64 check, see:

https://docs.python.org/3.11/library/binascii.html#binascii.a2b_base64

### euporie.core.terminal.edit_in_editor

euporie.core.terminal.**edit_in_editor**(*filename: str*, *line_number: int = 0*) → None

Suspend the current app and edit a file in an external editor.

### euporie.core.terminal.extend_enum

euporie.core.terminal.**extend_enum**(*enumeration*, *name*, **args*, ***kwds*)

Add a new member to an existing Enum.

### euporie.core.terminal.get_app

euporie.core.terminal.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.terminal.lru_cache

euporie.core.terminal.**lru_cache**(*maxsize=128*, *typed=False*)

Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

### euporie.core.terminal.passthrough

euporie.core.terminal.**passthrough**(*cmd: str*) → str

Wrap an escape sequence for terminal passthrough.

### euporie.core.terminal.register_bindings

euporie.core.terminal.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.terminal.run_in_terminal

euporie.core.terminal.**run_in_terminal**(*func: Callable[[], _T]*, *render_cli_done: bool = False*, *in_executor: bool = False*) → Awaitable[_T]

Run function on the terminal above the current application or prompt.

What this does is first hiding the prompt, then running this callable (which can safely output to the terminal), and then again rendering the prompt which causes the output of this function to scroll above the prompt.

`func` is supposed to be a synchronous function. If you need an asynchronous version of this function, use the `in_terminal` context manager directly.

**Parameters**

- **func** – The callable to execute.
- **render_cli_done** – When True, render the interface in the 'Done' state first, then execute the function. If False, erase the interface first.
- **in_executor** – When True, run in executor. (Use this for long blocking functions, when you don't want to block the event loop.)

**Returns**
A *Future*.

### Classes

| | |
|---|---|
| *ClipboardData*(input_, output, config) | A terminal query to retrieve clipboard contents. |
| *ColorDepth*(value[, names, module, qualname, ...]) | Possible color depth values for the output. |
| *Colors*(input_, output, config) | A terminal query to retrieve colours as hex codes. |
| *CsiUStatus*(input_, output, config) | A terminal query to check for CSI-u support. |
| *DepthOfColor*(input_, output, config) | Determine the suspected color depth of the terminal. |
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *ItermGraphicsStatus*(input_, output, config) | A terminal query to check for iTerm graphics support. |
| *KeyProcessor*(key_bindings) | Statemachine that receives `KeyPress` instances and according to the key bindings in the given `KeyBindings`, calls the matching handlers. |
| *Keys*(value[, names, module, qualname, type, ...]) | List of keys for use in key bindings. |
| *KittyGraphicsStatus*(input_, output, config) | A terminal query to check for kitty graphics support. |
| *PixelDimensions*(input_, output, config) | A terminal query to check the terminal's dimensions in pixels. |
| *SgrPixelStatus*(input_, output, config) | A terminal query to check for Pixel SGR support. |
| *SixelGraphicsStatus*(input_, output, config) | A terminal query to check for sixel graphics support. |
| *TerminalInfo*(input_, output, config) | A class to gather and hold information about the terminal. |
| *TerminalQuery*(input_, output, config) | A class representing a terminal query. |
| *dt* | alias of *datetime* |

### euporie.core.terminal.ClipboardData

**class** euporie.core.terminal.**ClipboardData**(*input_: Input*, *output: Output*, *config:* Config)

A terminal query to retrieve clipboard contents.

### euporie.core.terminal.ColorDepth

**class** euporie.core.terminal.**ColorDepth**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Possible color depth values for the output.

### euporie.core.terminal.Colors

**class** euporie.core.terminal.**Colors**(*input_: Input*, *output: Output*, *config:* Config)

A terminal query to retrieve colours as hex codes.

### euporie.core.terminal.CsiUStatus

**class** euporie.core.terminal.**CsiUStatus**(*input_: Input*, *output: Output*, *config:* Config)

A terminal query to check for CSI-u support.

### euporie.core.terminal.DepthOfColor

**class** euporie.core.terminal.**DepthOfColor**(*input_: Input*, *output: Output*, *config:* Config)

Determine the suspected color depth of the terminal.

### euporie.core.terminal.Event

**class** euporie.core.terminal.**Event**(*sender: _Sender*, *handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.terminal.ItermGraphicsStatus

**class** euporie.core.terminal.**ItermGraphicsStatus**(*input_: Input*, *output: Output*, *config:* Config)

A terminal query to check for iTerm graphics support.

### euporie.core.terminal.KeyProcessor

**class** euporie.core.terminal.**KeyProcessor**(*key_bindings:* KeyBindingsBase)

Statemachine that receives `KeyPress` instances and according to the key bindings in the given `KeyBindings`, calls the matching handlers.

```
p = KeyProcessor(key_bindings)

# Send keys into the processor.
p.feed(KeyPress(Keys.ControlX, ''))
p.feed(KeyPress(Keys.ControlC, '')

# Process all the keys in the queue.
p.process_keys()

# Now the ControlX-ControlC callback will be called if this sequence is
# registered in the key bindings.
```

**Parameters**

**key_bindings** – *KeyBindingsBase* instance.

### euporie.core.terminal.Keys

**class** euporie.core.terminal.**Keys**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

List of keys for use in key bindings.

Note that this is an "StrEnum", all values can be compared against strings.

### euporie.core.terminal.KittyGraphicsStatus

**class** euporie.core.terminal.**KittyGraphicsStatus**(*input_: Input*, *output: Output*, *config:* Config)

A terminal query to check for kitty graphics support.

### euporie.core.terminal.PixelDimensions

**class** euporie.core.terminal.**PixelDimensions**(*input_: Input*, *output: Output*, *config:* Config)

A terminal query to check the terminal's dimensions in pixels.

### euporie.core.terminal.SgrPixelStatus

**class** euporie.core.terminal.**SgrPixelStatus**(*input_: Input*, *output: Output*, *config:* Config)

A terminal query to check for Pixel SGR support.

### euporie.core.terminal.SixelGraphicsStatus

**class** euporie.core.terminal.**SixelGraphicsStatus**(*input_: Input*, *output: Output*, *config:* Config)

A terminal query to check for sixel graphics support.

### euporie.core.terminal.TerminalInfo

**class** euporie.core.terminal.**TerminalInfo**(*input_: Input*, *output: Output*, *config:* Config)

A class to gather and hold information about the terminal.

### euporie.core.terminal.TerminalQuery

**class** euporie.core.terminal.**TerminalQuery**(*input_: Input*, *output: Output*, *config:* Config)

A class representing a terminal query.

This allows a control sequence to sent to the terminal, the response interpreted, and the received value processed and stored.

### euporie.core.terminal.dt

euporie.core.terminal.**dt**

alias of *datetime*

**class** euporie.core.terminal.**ClipboardData**(*input_: Input*, *output: Output*, *config:* Config)

Bases: *TerminalQuery*

A terminal query to retrieve clipboard contents.

**await_response**(*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = False**

**cmd = '\x1b]52;c;?\x1b\\'**

**default: Any | None = ''**

**pattern: re.Pattern | None =**
**re.compile('^\\x1b\\]52;(?:c|p)?;(?P<data>[A-Za-z0-9+/=]+)\\x1b\\\\\\Z')**

**send**(*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**

The last value received, or the default value.

**verify**(*data: str*) → str

Verify the terminal responds.

**class** euporie.core.terminal.**Colors**(*input_: Input*, *output: Output*, *config:* Config)

Bases: *TerminalQuery*

A terminal query to retrieve colours as hex codes.

**await_response**(*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = True**

**cmd = '\x1b]10;?\x1b\\\x1b]11;?\x1b\\\x1b]4;0;?\x1b\\\x1b]4;1;?\x1b\\\x1b]4;2;?\x1b\\\x1b]4;3;?\x1b\\\x1b]4;4;?\x1b\\\x1b]4;5;?\x1b\\\x1b]4;6;?\x1b\\\x1b]4;7;?\x1b\\\x1b]4;8;?\x1b\\\x1b]4;9;?\x1b\\\x1b]4;10;?\x1b\\\x1b]4;11;?\x1b\\\x1b]4;12;?\x1b\\\x1b]4;13;?\x1b\\\x1b]4;14;?\x1b\\\x1b]4;15;?\x1b\\'**

**default: Any | None = {'ansiblack': '#000000', 'ansiblue': '#0d73cc', 'ansibrightblack': '#767676', 'ansibrightblue': '#1a8fff', 'ansibrightcyan': '#14ffff', 'ansibrightgreen': '#23fd00', 'ansibrightmagenta': '#fd28ff', 'ansibrightpurple': '#fd28ff', 'ansibrightred': '#f2201f', 'ansibrightwhite': '#ffffff', 'ansibrightyellow': '#fffd00', 'ansicyan': '#0dcdcd', 'ansigray': '#767676', 'ansigreen': '#19cb00', 'ansimagenta': '#cb1ed1', 'ansipurple': '#9841bb', 'ansired': '#cc0403', 'ansiwhite': '#dddddd', 'ansiyellow': '#cecb00', 'bg': '#232627', 'fg': '#fcfcfc'}**

**pattern: re.Pattern | None = re.compile('^\\x1b\\](?P<c>(\\d+;)?\\d+)+;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|\\0x7)')**

**send**(*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**

The last value received, or the default value.

**verify**(*data: str*) → dict[str, str]

Verify the response contains a colour.

**class** euporie.core.terminal.**CsiUStatus**(*input_: Input*, *output: Output*, *config:* Config)

Bases: *TerminalQuery*

A terminal query to check for CSI-u support.

**await_response**(*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = True**

**cmd = '\x1b[?u'**

**default: Any | None = False**

**pattern: re.Pattern | None = re.compile('^\\x1b\\[\\?\\d+u')**

**send** (*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**
The last value received, or the default value.

**verify** (*data: str*) → bool

Verify the terminal responds.

**class** euporie.core.terminal.**DepthOfColor** (*input_: Input*, *output: Output*, *config:* Config)

Bases: *TerminalQuery*

Determine the suspected color depth of the terminal.

**await_response** (*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = False**

**cmd = ''**

**default: Any | None = 'DEPTH_24_BIT'**

**pattern: re.Pattern | None = None**

**send** (*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**
The last value received, or the default value.

**verify** (*data: str*) → Any | None

Verify the response from the terminal.

**class** euporie.core.terminal.**ItermGraphicsStatus** (*input_: Input*, *output: Output*, *config:* Config)

Bases: *TerminalQuery*

A terminal query to check for iTerm graphics support.

**await_response** (*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = True**

**cmd = '\x1b[>q'**

**default: Any | None = False**

**pattern: re.Pattern | None =**
**re.compile('^\\x1bP>\\|(?P<term>[^\\x1b]+)\\x1b\\\\')**

**send** (*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**
The last value received, or the default value.

**verify**(*data: str*) → bool

Verify iterm graphics are supported by the terminal.

**class** euporie.core.terminal.**KittyGraphicsStatus**(*input_: Input*, *output: Output*, *config:* Config)

Bases: *TerminalQuery*

A terminal query to check for kitty graphics support.

**await_response**(*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = True**

**cmd = '\x1b_Gi=4294967295,s=1,v=1,a=q,t=d,f=24;AAAA\x1b\\'**

**default: Any | None = False**

**pattern: re.Pattern | None =**
**re.compile('^\\x1b_Gi=(4294967295|0);(?P<status>OK)\\x1b\\\\\\Z')**

**send**(*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**
The last value received, or the default value.

**verify**(*data: str*) → bool

Verify the terminal response means kitty graphics are supported.

**class** euporie.core.terminal.**PixelDimensions**(*input_: Input*, *output: Output*, *config:* Config)

Bases: *TerminalQuery*

A terminal query to check the terminal's dimensions in pixels.

**await_response**(*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = True**

**cmd = '\x1b[14t'**

**default: Any | None = (0, 0)**

**pattern: re.Pattern | None =**
**re.compile('^\\x1b\\[4;(?P<y>\\d+);(?P<x>\\d+)t')**

**send**(*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**
The last value received, or the default value.

**verify**(*data: str*) → tuple[int, int] | None

Verify the terminal responded with pixel dimensions.

**class** euporie.core.terminal.**SgrPixelStatus**(*input_: Input*, *output: Output*, *config:* Config)

Bases: *TerminalQuery*

A terminal query to check for Pixel SGR support.

**await_response**(*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = True**

**cmd = '\x1b[?1016h\x1b[?1016$p\x1b[?1016l'**

**default: Any | None = False**

**pattern: re.Pattern | None =**
**re.compile('^\\x1b\\[\\?1016;(?P<Pm>\\d)\\$\\Z')**

**send**(*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**
The last value received, or the default value.

**verify**(*data: str*) → bool

Verify the terminal response means SGR pixel-mode is supported.

**class** euporie.core.terminal.**SixelGraphicsStatus**(*input_: Input*, *output: Output*, *config:* Config)

Bases: *TerminalQuery*

A terminal query to check for sixel graphics support.

**await_response**(*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = True**

**cmd = '\x1b[c'**

**default: Any | None = False**

**pattern: re.Pattern | None =**
**re.compile('^\\x1b\\[\\?(?:\\d+;)*(?P<sixel>4)(?:;\\d+)*c\\Z')**

**send**(*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**
The last value received, or the default value.

**verify**(*data: str*) → bool

Verify the terminal response means sixel graphics are supported.

**class** euporie.core.terminal.**TerminalInfo**(*input_: Input*, *output: Output*, *config:* Config)

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[*euporie.core.terminal.TerminalQuery*]*) → *TerminalQuery*

Instantiate and registers a query's response with the input parser.

**send_all**() → None

Send the command for all queries.

**property terminal_size_px: tuple[int, int]**

Get the pixel dimensions of the terminal.

**class** euporie.core.terminal.**TerminalQuery**(*input_: Input*, *output: Output*, *config:* Config)

Bases: object

A class representing a terminal query.

This allows a control sequence to sent to the terminal, the response interpreted, and the received value processed and stored.

**await_response**(*timeout: float = 0.2*) → bool

Wait for a response from the terminal.

**cache = False**

**cmd = ''**

**default: Any | None = None**

**pattern: re.Pattern | None = None**

**send**(*flush: bool = True*) → None

Send the terminal query command to the output.

**property value: Any**

Return the last known value for the query.

**Returns**
The last value received, or the default value.

**verify**(*data: str*) → Any | None

Verify the response from the terminal.

euporie.core.terminal.**edit_in_editor**(*filename: str*, *line_number: int = 0*) → None

Suspend the current app and edit a file in an external editor.

euporie.core.terminal.**passthrough**(*cmd: str*) → str

Wrap an escape sequence for terminal passthrough.

### euporie.core.utils

Miscellaneou utility classes.

### Functions

| | |
|---|---|
| *dict_merge*(target_dict, input_dict) | Merge the second dictionary onto the first. |
| *on_click*(func) | Return a mouse handler which call a given function on click. |
| *overload*(func) | Decorator for overloaded functions/methods. |
| *run_in_thread_with_context*(func, *args[, daemon]) | Run a function in an thread, but make sure it uses the same contextvars. |

### euporie.core.utils.dict_merge

euporie.core.utils.**dict_merge**(*target_dict: dict*, *input_dict: dict*) → None

Merge the second dictionary onto the first.

### euporie.core.utils.on_click

euporie.core.utils.**on_click**(*func: Callable*) → MouseHandler

Return a mouse handler which call a given function on click.

### euporie.core.utils.overload

euporie.core.utils.**overload**(*func*)

Decorator for overloaded functions/methods.

In a stub file, place two or more stub definitions for the same function in a row, each decorated with @overload.

For example:

```
@overload
def utf8(value: None) -> None: ...
@overload
def utf8(value: bytes) -> bytes: ...
@overload
def utf8(value: str) -> bytes: ...
```

In a non-stub file (i.e. a regular .py file), do the same but follow it with an implementation. The implementation should *not* be decorated with @overload:

```
@overload
def utf8(value: None) -> None: ...
@overload
def utf8(value: bytes) -> bytes: ...
@overload
def utf8(value: str) -> bytes: ...
def utf8(value):
    ...  # implementation goes here
```

The overloads for a function can be retrieved at runtime using the get_overloads() function.

### euporie.core.utils.run_in_thread_with_context

euporie.core.utils.**run_in_thread_with_context**(*func: Callable*, **args: Any*, *daemon: bool = True*, ***kwargs: Any*) → None

Run a function in an thread, but make sure it uses the same contextvars.

This is required so that the function will see the right application.

### Classes

| | |
|---|---|
| *ChainedList*(*lists) | A list-like class which chains multiple lists. |
| *MouseButton*(value[, names, module, ...]) | |
| *MouseEventType*(value[, names, module, ...]) | |
| *Thread*([group, target, name, args, kwargs, ...]) | A class that represents a thread of control. |
| *TypeVar* | Type variable. |
| *chain* | chain(*iterables) --> chain object |

### euporie.core.utils.ChainedList

**class** euporie.core.utils.**ChainedList**(**lists: Iterable[T]*)

A list-like class which chains multiple lists.

### euporie.core.utils.MouseButton

**class** euporie.core.utils.**MouseButton**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.utils.MouseEventType

**class** euporie.core.utils.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.utils.Thread

**class** euporie.core.utils.**Thread**(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs=None*, ***, *daemon=None*)

A class that represents a thread of control.

This class can be safely subclassed in a limited fashion. There are two ways to specify the activity: by passing a callable object to the constructor, or by overriding the run() method in a subclass.

### euporie.core.utils.TypeVar

**class** euporie.core.utils.**TypeVar**

Type variable.

The preferred way to construct a type variable is via the dedicated syntax for generic functions, classes, and type aliases:

```
class Sequence[T]:  # T is a TypeVar
    ...
```

This syntax can also be used to create bound and constrained type variables:

```
# S is a TypeVar bound to str
class StrSequence[S: str]:
    ...

# A is a TypeVar constrained to str or bytes
class StrOrBytesSequence[A: (str, bytes)]:
    ...
```

However, if desired, reusable type variables can also be constructed manually, like so:

```
T = TypeVar('T')  # Can be anything
S = TypeVar('S', bound=str)  # Can be any subtype of str
A = TypeVar('A', str, bytes)  # Must be exactly str or bytes
```

Type variables exist primarily for the benefit of static type checkers. They serve as the parameters for generic types as well as for generic function and type alias definitions.

The variance of type variables is inferred by type checkers when they are created through the type parameter syntax and when `infer_variance=True` is passed. Manually created type variables may be explicitly marked covariant or contravariant by passing `covariant=True` or `contravariant=True`. By default, manually created type variables are invariant. See PEP 484 and PEP 695 for more details.

### euporie.core.utils.chain

**class** euporie.core.utils.**chain**

chain(*iterables) –> chain object

Return a chain object whose .__next__() method returns elements from the first iterable until it is exhausted, then elements from the next iterable, until all of the iterables are exhausted.

**class** euporie.core.utils.**ChainedList**(**lists: Iterable[T]*)

Bases: Sequence[T]

A list-like class which chains multiple lists.

**count**(*value*) → integer -- return number of occurrences of value

**property data: list[T]**

Return the list data.

**index**(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

euporie.core.utils.**dict_merge**(*target_dict: dict*, *input_dict: dict*) → None

Merge the second dictionary onto the first.

euporie.core.utils.**on_click**(*func: Callable*) → MouseHandler

Return a mouse handler which call a given function on click.

euporie.core.utils.**run_in_thread_with_context**(*func: Callable*, **args: Any*, *daemon: bool = True*, ***kwargs: Any*) → None

Run a function in an thread, but make sure it uses the same contextvars.

This is required so that the function will see the right application.

### euporie.core.validation

Custom validators.

### Classes

| | |
|---|---|
| *KernelValidator*(kernel) | Validate kernel input using a kernel code completeness call. |
| *Validator*() | Abstract base class for an input validator. |

### euporie.core.validation.KernelValidator

**class** euporie.core.validation.**KernelValidator**(*kernel: Kernel*)

Validate kernel input using a kernel code completeness call.

### euporie.core.validation.Validator

**class** euporie.core.validation.**Validator**

Abstract base class for an input validator.

A validator is typically created in one of the following two ways:

- Either by overriding this class and implementing the *validate* method.
- Or by passing a callable to *Validator.from_callable*.

If the validation takes some time and needs to happen in a background thread, this can be wrapped in a `ThreadedValidator`.

### Exceptions

| | |
|---|---|
| *ValidationError*([cursor_position, message]) | Error raised by `Validator.validate()`. |

### euporie.core.validation.ValidationError

**exception** euporie.core.validation.**ValidationError**(*cursor_position: int = 0*, *message: str = ''*)

Error raised by `Validator.validate()`.

**Parameters**

- **cursor_position** – The cursor position where the error occurred.
- **message** – Text.

**class** euporie.core.validation.**KernelValidator**(*kernel:* Kernel)

Bases: *Validator*

Validate kernel input using a kernel code completeness call.

**classmethod from_callable**(*validate_func: Callable[[str], bool]*, *error_message: str = 'Invalid input'*, *move_cursor_to_end: bool = False*) → *Validator*

Create a validator from a simple validate callable. E.g.:

```
def is_valid(text):
    return text in ['hello', 'world']
Validator.from_callable(is_valid, error_message='Invalid input')
```

**Parameters**

- **validate_func** – Callable that takes the input string, and returns *True* if the input is valid input.
- **error_message** – Message to be displayed if the input is invalid.
- **move_cursor_to_end** – Move the cursor to the end of the input, if the input is invalid.

**validate**(*document:* Document) → None

Validate the input synchronously.

**async validate_async**(*document:* Document) → None

Return a *Future* which is set when the validation is ready.

### euporie.core.widgets

Contain various widgets used in euporie.core.

### Modules

| | |
|---|---|
| *euporie.core.widgets.cell* | Define a cell object with input are and rich outputs, and related objects. |
| *euporie.core.widgets.cell_outputs* | Contain a container for the cell output area. |
| *euporie.core.widgets.decor* | Decorative widgets. |
| *euporie.core.widgets.dialog* | Dialog. |
| *euporie.core.widgets.display* | Define custom controls which re-render on resize. |
| *euporie.core.widgets.file_browser* | Define a file browser widget. |
| *euporie.core.widgets.formatted_text_area* | Contain dpdated ANSI parsing and Formatted Text processing. |
| *euporie.core.widgets.forms* | Contain input widgets. |
| *euporie.core.widgets.inputs* | Define a cell object with input are and rich outputs, and related objects. |
| *euporie.core.widgets.layout* | Define widget for defining layouts. |
| *euporie.core.widgets.menu* | Define an application menu. |
| *euporie.core.widgets.pager* | Contain a container for the cell output area. |
| *euporie.core.widgets.palette* | Contain the command palette container. |
| *euporie.core.widgets.search* | Define the global search toolbar and related search functions. |
| *euporie.core.widgets.status* | Define a status-bar widget. |
| *euporie.core.widgets.tree* | Define a tree-view widget. |

### euporie.core.widgets.cell

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

### Functions

| | |
|---|---|
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *cast*(typ, val) | Cast a value to a type. |
| *get_app*() | Get the current active (running) Application. |
| *get_cell_id*(cell_json) | Return the cell ID field defined in a cell JSON object. |
| *on_click*(func) | Return a mouse handler which call a given function on click. |

### euporie.core.widgets.cell.add_setting

euporie.core.widgets.cell.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.widgets.cell.cast

euporie.core.widgets.cell.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.widgets.cell.get_app

euporie.core.widgets.cell.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.cell.get_cell_id

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

Return 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

### euporie.core.widgets.cell.on_click

euporie.core.widgets.cell.**on_click**(*func: Callable*) → MouseHandler

Return a mouse handler which call a given function on click.

### Classes

| | |
|---|---|
| *Cell*(index, json, kernel_tab[, is_new]) | A kernel_tab cell element. |
| *CellOutputArea*(json, parent[, style]) | An area below a cell where one or more cell outputs can be shown. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *Container*() | Base class for user interface layout. |
| *DeduplicateCompleter*(completer) | Asynchronous wrapper around a completer that removes duplicates. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *Document*([text, cursor_position, selection]) | This is a immutable class around the text and cursor position, and contains methods for querying this data, e.g. |
| *DynamicCompleter*(get_completer) | Completer class that can dynamically returns any Completer. |
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *FirstInspector*(inspectors) | Return results of the first inspector to response. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *KernelInput*(kernel_tab[, text, multiline, ...]) | Kernel input text areas. |
| *LspCell*(id, idx, path, kind, language, text, ...) | An LSP client's representation of a cell. |
| *LspCompleter*(lsp, path) | A completer for documents using an LSP. |
| *LspFormatter*(lsp, path[, languages]) | Format a document using a LSP server. |
| *LspInspector*(lsp, path) | Inspector which retrieves contextual help from a Language Server. |
| *Path*(*args, **kwargs) | PurePath subclass that can make system calls. |
| *Report*([iterable]) | Class for storing a diagnostic report. |
| *StdInput*(kernel_tab) | A widget to accept kernel input. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *WeakKeyDictionary*([dict]) | Mapping class that references keys weakly. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.widgets.cell.Cell

**class** euporie.core.widgets.cell.**Cell**(*index: int*, *json: dict*, *kernel_tab:* BaseNotebook, *is_new: bool = False*)

A kernel_tab cell element.

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

### euporie.core.widgets.cell.CellOutputArea

**class** euporie.core.widgets.cell.**CellOutputArea**(*json: list[dict[str, Any]]*, *parent: OutputParent | None*, *style: str = ''*)

An area below a cell where one or more cell outputs can be shown.

### euporie.core.widgets.cell.Condition

**class** euporie.core.widgets.cell.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.cell.ConditionalContainer

**class** euporie.core.widgets.cell.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.cell.Container

**class** euporie.core.widgets.cell.**Container**

Base class for user interface layout.

### euporie.core.widgets.cell.DeduplicateCompleter

**class** euporie.core.widgets.cell.**DeduplicateCompleter**(*completer:* Completer)

Asynchronous wrapper around a completer that removes duplicates.

Only the first unique completions are kept. Completions are considered to be a duplicate if they result in the same document text when they would be applied.

### euporie.core.widgets.cell.Dimension

**class** euporie.core.widgets.cell.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.widgets.cell.Document

**class** euporie.core.widgets.cell.**Document**(*text: str = ''*, *cursor_position: int | None = None*, *selection: prompt_toolkit.selection.SelectionState | None = None*)

This is a immutable class around the text and cursor position, and contains methods for querying this data, e.g. to give the text before the cursor.

This class is usually instantiated by a *Buffer* object, and accessed as the *document* property of that class.

**Parameters**

- **text** – string
- **cursor_position** – int
- **selection** – *SelectionState*

### euporie.core.widgets.cell.DynamicCompleter

**class** euporie.core.widgets.cell.**DynamicCompleter**(*get_completer: Callable[[], prompt_toolkit.completion.base.Completer | None]*)

Completer class that can dynamically returns any Completer.

**Parameters**

**get_completer** – Callable that returns a *Completer* instance.

### euporie.core.widgets.cell.Event

**class** euporie.core.widgets.cell.**Event**(*sender: _Sender*, *handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.widgets.cell.FirstInspector

**class** euporie.core.widgets.cell.**FirstInspector**(*inspectors: Sequence[Inspector] | Callable[[], Sequence[Inspector]]*)

Return results of the first inspector to response.

### euporie.core.widgets.cell.FormattedTextControl

**class** euporie.core.widgets.cell.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

> The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That

handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or `Filter`: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **key_bindings** – a `KeyBindings` object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.widgets.cell.HSplit

**class** euporie.core.widgets.cell.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: VerticalAlign* = *VerticalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.cell.KernelInput

**class** euporie.core.widgets.cell.**KernelInput**(*kernel_tab: KernelTab*, *text: str = ''*, *multiline: FilterOrBool = True*, *password: FilterOrBool = False*, *lexer: Lexer | None = None*, *auto_suggest: AutoSuggest | None = None*, *completer: Completer | None = None*, *complete_while_typing: FilterOrBool = True*, *validator: Validator | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *history: History | None = None*, *focusable: FilterOrBool = True*, *focus_on_click: FilterOrBool = True*, *wrap_lines: FilterOrBool = False*, *read_only: FilterOrBool = False*, *width: AnyDimension = None*, *height: AnyDimension = None*, *dont_extend_height: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *line_numbers: bool = False*, *get_line_prefix: GetLinePrefixCallable | None = None*, *scrollbar: FilterOrBool = True*, *style: str = 'class:kernel-input'*, *search_field: SearchToolbar | None = None*, *preview_search: FilterOrBool = False*, *prompt: AnyFormattedText = ''*, *input_processors: list[Processor] | None = None*, *name: str = ''*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *on_cursor_position_changed: Callable[[Buffer], None] | None = None*, *tempfile_suffix: str | Callable[[], str] = ''*, *key_bindings: KeyBindingsBase | None = None*, *enable_history_search: FilterOrBool = False*, *autosuggest_while_typing: FilterOrBool = True*, *validate_while_typing: FilterOrBool = False*, *scroll_offsets: ScrollOffsets | None = None*, *formatters: list[Formatter] | None = None*, *language: str | Callable[[], str] | None = None*, *diagnostics: Report | Callable[[], Report] | None = None*, *inspector: Inspector | None = None*, *show_diagnostics: FilterOrBool = True*)

Kernel input text areas.

A customized text area for the cell input.

### euporie.core.widgets.cell.LspCell

**class** euporie.core.widgets.cell.**LspCell**(*id: str*, *idx: int*, *path: Path*, *kind: str*, *language: str*, *text: str*, *execution_count: int*, *metadata: dict[str, Any] | None = None*)

An LSP client's representation of a cell.

### euporie.core.widgets.cell.LspCompleter

**class** euporie.core.widgets.cell.**LspCompleter**(*lsp:* LspClient, *path:* Path)

A completer for documents using an LSP.

### euporie.core.widgets.cell.LspFormatter

**class** euporie.core.widgets.cell.**LspFormatter**(*lsp:* LspClient, *path:* Path, *languages: set[str]* | *None* = *None*)

Format a document using a LSP server.

### euporie.core.widgets.cell.LspInspector

**class** euporie.core.widgets.cell.**LspInspector**(*lsp:* LspClient, *path:* Path)

Inspector which retrieves contextual help from a Language Server.

### euporie.core.widgets.cell.Path

**class** euporie.core.widgets.cell.**Path**(**args*, ***kwargs*)

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

### euporie.core.widgets.cell.Report

**class** euporie.core.widgets.cell.**Report**(*iterable=()*, */*)

Class for storing a diagnostic report.

### euporie.core.widgets.cell.StdInput

**class** euporie.core.widgets.cell.**StdInput**(*kernel_tab:* KernelTab)

A widget to accept kernel input.

### euporie.core.widgets.cell.VSplit

**class** euporie.core.widgets.cell.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: HorizontalAlign* = *HorizontalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.cell.WeakKeyDictionary

**class** euporie.core.widgets.cell.**WeakKeyDictionary**(*dict=None*)

Mapping class that references keys weakly.

Entries in the dictionary will be discarded when there is no longer a strong reference to the key. This can be used to associate additional data with an object owned by other parts of an application without adding attributes to those objects. This can be especially useful with objects that override attribute accesses.

### euporie.core.widgets.cell.Window

**class** euporie.core.widgets.cell.**Window**(*content: UIControl | None = None, width: AnyDimension = None, height: AnyDimension = None, z_index: int | None = None, dont_extend_width: FilterOrBool = False, dont_extend_height: FilterOrBool = False, ignore_content_width: FilterOrBool = False, ignore_content_height: FilterOrBool = False, left_margins: Sequence[Margin] | None = None, right_margins: Sequence[Margin] | None = None, scroll_offsets: ScrollOffsets | None = None, allow_scroll_beyond_bottom: FilterOrBool = False, wrap_lines: FilterOrBool = False, get_vertical_scroll: Callable[[Window], int] | None = None, get_horizontal_scroll: Callable[[Window], int] | None = None, always_hide_cursor: FilterOrBool = False, cursorline: FilterOrBool = False, cursorcolumn: FilterOrBool = False, colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None, align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT, style: str | Callable[[], str] = '', char: None | str | Callable[[], str] = None, get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.core.widgets.cell.partial

**class** euporie.core.widgets.cell.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.widgets.cell.**Cell**(*index: int, json: dict, kernel_tab: BaseNotebook, is_new: bool = False*)

Bases: object

A kernel_tab cell element.

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

**add_output**(*output_json: dict[str, Any]*) → None

Add a new output to the cell.

**property cell_type: str**

Determine the current cell type.

**clear_output**(*wait: bool = False*) → None

Remove the cells output, optionally when new output is generated.

**close**() → None

Signal that the cell is no longer present in the notebook.

**async edit_in_editor**() → None

Edit the cell in $EDITOR.

**property execution_count: int**

Retrieve the execution count from the cell's JSON.

**focus**(*position: int | None = None*, *scroll: bool = False*) → None

Focus the relevant control in this cell.

**Parameters**

- **position** – An optional cursor position index to apply to the input box
- **scroll** – Whether to scroll the page to make the selection visible

**property focused: bool**

Determine if the cell currently has focus.

**get_input**(*prompt: str = 'Please enter a value:'*, *password: bool = False*) → None

Scroll the cell requesting input into view and render it before asking for input.

**hide_input**() → None

Set the cell inputs to visible.

**hide_output**() → None

Set the cell outputs to visible.

**property id: str**

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

**property input: str**

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

**input_box: *KernelInput***

**property language: str**

Return the cell's code language.

**async load_lsps**() → None

Add hooks to the notebook LSP client.

**property lsp_cell: *LspCell***

Capture the cell's attributes for an LSP server.

**lsp_change_handler**(*lsp:* LspClient) → None

Tell the LSP server a cell has changed.

**lsp_close_handler**(*lsp:* LspClient) → None

Notify the LSP of a deleted cell.

**lsp_open_handler**(*lsp:* LspClient) → None

If the LSP does not support notebooks, open this cell as a text document.

**lsp_update_diagnostics**(*lsp:* LspClient) → None

Process a new diagnostic report from the LSP.

**property output_json: list[dict[str, Any]]**

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

**property path: *Path***

Return a virtual path for this cell (used by LSP clients).

Pylance appears to use URIs like: `../folder.notebook.ipynb:pylance-notebook-cell:W0sZmlsZQ==.py`, which I've emulated here.

**property prompt: str**

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

**ran**(*content: dict | None = None*) → None

Update the cell status and update display when the cell has finished.

**refresh**(*now: bool = True*) → None

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

**remove_outputs**() → None

Remove all outputs from the cell.

**report**() → *Report*

Return the current diagnostic reports.

**run_or_render**(*buffer:* Buffer *| None = None*, *wait: bool = False*, *callback: Callable[..., None] | None = None*) → bool

Send the cell's source code the the kernel to run.

**Parameters**

- **buffer** – Unused parameter, required when accepting the contents of a cell's input buffer
- **wait** – Has no effect
- **callback** – Callable to run when the kernel has finished running the cell

**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

**set_execution_count**(*n: int*) → None

Set the execution count of the cell.

**set_metadata**(*path: tuple[str, ...]*, *data: Any*) → None

Set a value in the metadata at an arbitrary path.

**Parameters**

- **path** – A tuple of path level names to create
- **data** – The value to add

**set_status**(*status: str*) → None

Set the execution status of the cell.

**show_input**() → None

Set the cell inputs to visible.

**show_output**() → None

Set the cell outputs to visible.

**property suffix: str**

Return the file suffix matching the current cell type.

**toggle_input**() → None

Toggle the visibility of the cell input.

**toggle_output**() → None

Toggle the visibility of the cell outputs.

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

Return 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

### euporie.core.widgets.cell_outputs

Contain a container for the cell output area.

### Functions

| | |
|---|---|
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *find_route*(from_, to) | Find the shortest conversion path between two formats. |
| *get_app*() | Get the current active (running) Application. |
| *to_container*(container) | Monkey-patch *to_container* to collect container status functions. |

### euporie.core.widgets.cell_outputs.abstractmethod

euporie.core.widgets.cell_outputs.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):

…

### euporie.core.widgets.cell_outputs.add_setting

euporie.core.widgets.cell_outputs.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[*Setting*], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.widgets.cell_outputs.find_route

euporie.core.widgets.cell_outputs.**find_route**(*from_: str*, *to: str*) → list | None

Find the shortest conversion path between two formats.

### euporie.core.widgets.cell_outputs.get_app

euporie.core.widgets.cell_outputs.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.cell_outputs.to_container

euporie.core.widgets.cell_outputs.**to_container**(*container: AnyContainer*) → *Container*

Monkey-patch *to_container* to collect container status functions.

### Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *Box*(body[, padding, padding_left, ...]) | Add padding around a container. |
| *CellOutput*(json, parent) | Represent a single cell output. |
| *CellOutputArea*(json, parent[, style]) | An area below a cell where one or more cell outputs can be shown. |
| *CellOutputDataElement*(mime, data, metadata, ...) | A cell output element which display data. |
| *CellOutputElement*(mime, data, metadata, parent) | Base class for the various types of cell outputs (display data or widgets). |
| *CellOutputJsonElement*(mime, data, metadata, ...) | A cell output element which displays JSON. |
| *CellOutputWidgetElement*(mime, data, ...) | A cell output element which displays ipywidgets. |
| *Datum*(data, *args, **kwargs) | Class for storing and converting display data. |
| *Display*(datum[, height, width, focusable, ...]) | Rich output displays. |
| *DynamicContainer*(get_container) | Container class that dynamically returns any Container. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *JsonView*(data[, title, expanded]) | A JSON-view container. |
| *PurePath*(*args, **kwargs) | Base class for manipulating paths without I/O. |
| *SimpleCache*([maxsize]) | Very simple cache that discards the oldest item when the cache size is exceeded. |

#### euporie.core.widgets.cell_outputs.ABCMeta

**class** euporie.core.widgets.cell_outputs.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

#### euporie.core.widgets.cell_outputs.Box

**class** euporie.core.widgets.cell_outputs.**Box**(*body: AnyContainer*, *padding: AnyDimension = None*, *padding_left: AnyDimension = None*, *padding_right: AnyDimension = None*, *padding_top: AnyDimension = None*, *padding_bottom: AnyDimension = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *char: None | str | Callable[[], str] = None*, *modal: bool = False*, *key_bindings: KeyBindings | None = None*)

Add padding around a container.

This also makes sure that the parent can provide more space than required by the child. This is very useful when wrapping a small element with a fixed size into a `VSplit` or `HSplit` object. The `HSplit` and `VSplit` try to make sure to adapt respectively the width and height, possibly shrinking other elements. Wrapping something in a `Box` makes it flexible.

**Parameters**

- **body** – Another container object.
- **padding** – The margin to be used around the body. This can be
- **padding_left** (*overridden by*) – *padding_bottom*.
- **padding_right`** – *padding_bottom*.
- **and** (*padding_top*) – *padding_bottom*.
- **style** – A style string.
- **char** – Character to be used for filling the space around the body. (This is supposed to be a character with a terminal width of 1.)

### euporie.core.widgets.cell_outputs.CellOutput

**class** euporie.core.widgets.cell_outputs.**CellOutput**(*json: dict[str, Any]*, *parent: OutputParent | None*)

Represent a single cell output.

Capable of displaying multiple mime representations of the same data.

TODO - allow the visible mime-type to be rotated.

### euporie.core.widgets.cell_outputs.CellOutputArea

**class** euporie.core.widgets.cell_outputs.**CellOutputArea**(*json: list[dict[str, Any]]*, *parent: OutputParent | None*, *style: str = ''*)

An area below a cell where one or more cell outputs can be shown.

### euporie.core.widgets.cell_outputs.CellOutputDataElement

**class** euporie.core.widgets.cell_outputs.**CellOutputDataElement**(*mime: str*, *data: Any*, *metadata: dict*, *parent: OutputParent | None*)

A cell output element which display data.

### euporie.core.widgets.cell_outputs.CellOutputElement

**class** euporie.core.widgets.cell_outputs.**CellOutputElement**(*mime: str*, *data: Any*, *metadata: dict*, *parent: OutputParent | None*)

Base class for the various types of cell outputs (display data or widgets).

### euporie.core.widgets.cell_outputs.CellOutputJsonElement

**class** euporie.core.widgets.cell_outputs.**CellOutputJsonElement**(*mime: str*, *data: dict[str, Any]*, *metadata: dict*, *parent: OutputParent | None*)

A cell output element which displays JSON.

### euporie.core.widgets.cell_outputs.CellOutputWidgetElement

**class** euporie.core.widgets.cell_outputs.**CellOutputWidgetElement**(*mime: str*, *data: dict[str, Any]*, *metadata: dict*, *parent: OutputParent | None*)

A cell output element which displays ipywidgets.

### euporie.core.widgets.cell_outputs.Datum

**class** euporie.core.widgets.cell_outputs.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Class for storing and converting display data.

### euporie.core.widgets.cell_outputs.Display

**class** euporie.core.widgets.cell_outputs.**Display**(*datum: Datum*, *height: AnyDimension = None*, *width: AnyDimension = None*, *focusable: FilterOrBool = False*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *always_hide_cursor: FilterOrBool = True*, *scrollbar: FilterOrBool = True*, *scrollbar_autohide: FilterOrBool = True*, *dont_extend_height: FilterOrBool = True*, *dont_extend_width: FilterOrBool = False*, *style: str | Callable[[], str] = ''*)

Rich output displays.

A container for displaying rich output data.

### euporie.core.widgets.cell_outputs.DynamicContainer

**class** euporie.core.widgets.cell_outputs.**DynamicContainer**(*get_container: Callable[[], AnyContainer]*)

Container class that dynamically returns any Container.

**Parameters**

**get_container** – Callable that returns a *Container* instance or any widget with a `__pt_container__` method.

### euporie.core.widgets.cell_outputs.HSplit

**class** euporie.core.widgets.cell_outputs.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.cell_outputs.JsonView

**class** euporie.core.widgets.cell_outputs.**JsonView**(*data: Any*, *title: str | None = None*, *expanded: bool = True*)

A JSON-view container.

### euporie.core.widgets.cell_outputs.PurePath

**class** euporie.core.widgets.cell_outputs.**PurePath**(**args*, ***kwargs*)

Base class for manipulating paths without I/O.

PurePath represents a filesystem path and offers operations which don't imply any actual filesystem I/O. Depending on your system, instantiating a PurePath will return either a PurePosixPath or a PureWindowsPath object. You can also instantiate either of these classes directly, regardless of your system.

### euporie.core.widgets.cell_outputs.SimpleCache

**class** euporie.core.widgets.cell_outputs.**SimpleCache**(*maxsize: int = 8*)

Very simple cache that discards the oldest item when the cache size is exceeded.

**Parameters**

**maxsize** – Maximum size of the cache. (Don't make it too big.)

**class** euporie.core.widgets.cell_outputs.**CellOutput**(*json: dict[str, Any]*, *parent: OutputParent | None*)

Bases: object

Represent a single cell output.

Capable of displaying multiple mime representations of the same data.

TODO - allow the visible mime-type to be rotated.

**property data: dict[str, Any]**

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.

**property element: *CellOutputElement***

Get the element for the currently selected mime type.

**get_element**(*mime: str*) → *CellOutputElement*

Return the currently displayed cell element.

**make_element**(*mime: str*) → *CellOutputElement*

Create a container for the cell output mime-type if it doesn't exist.

**Parameters**

**mime** – The mime-type for which to create an output element

**Returns**

A `OutputElement` container for the currently selected mime-type.

**scroll_left**() → None

Scroll the currently visible output left.

**scroll_right**() → None

Scroll the currently visible output right.

**property selected_mime: str**

Return the selected mime-type, selecting the first by default.

**update**() → None

Update the output by updating all child containers.

**class** euporie.core.widgets.cell_outputs.**CellOutputArea**(*json: list[dict[str, Any]]*, *parent: OutputParent | None*, *style: str = ''*)

Bases: `object`

An area below a cell where one or more cell outputs can be shown.

**add_output**(*output_json: dict[str, Any]*, *refresh: bool = True*) → None

Add a new output to the output area.

**property json: Any**

Return the control's display data.

**output_cache: *SimpleCache*[str, *CellOutput*] =**
**<prompt_toolkit.cache.SimpleCache object>**

**reset**() → None

Clear all outputs from the output area.

**scroll_left**() → None

Scroll the outputs left.

**scroll_right**() → None

Scroll the outputs right.

**to_plain_text**() → str

Convert the contents of the output to plain text.

**update**() → None

Update all existing outputs.

**class** euporie.core.widgets.cell_outputs.**CellOutputDataElement**(*mime: str*, *data: Any*, *metadata: dict*, *parent: OutputParent | None*)

Bases: *CellOutputElement*

A cell output element which display data.

**property data: Any**

Return the control's display data.

**scroll_left**() → None

Scroll the output left.

**scroll_right**() → None

Scroll the output right.

**class** euporie.core.widgets.cell_outputs.**CellOutputElement**(*mime: str*, *data: Any*, *metadata: dict*, *parent: OutputParent | None*)

Bases: object

Base class for the various types of cell outputs (display data or widgets).

**data: Any**

**scroll_left**() → None

Scroll the output left.

**scroll_right**() → None

Scroll the output right.

**class** euporie.core.widgets.cell_outputs.**CellOutputJsonElement**(*mime: str*, *data: dict[str, Any]*, *metadata: dict*, *parent: OutputParent | None*)

Bases: *CellOutputElement*

A cell output element which displays JSON.

**data: Any**

**scroll_left**() → None

Scroll the output left.

**scroll_right**() → None

Scroll the output right.

**class** euporie.core.widgets.cell_outputs.**CellOutputWidgetElement**(*mime: str*, *data: dict[str, Any]*, *metadata: dict*, *parent: OutputParent | None*)

Bases: *CellOutputElement*

A cell output element which displays ipywidgets.

**data: Any**

**scroll_left**() → None
Scroll the output left.

**scroll_right**() → None
Scroll the output right.

### euporie.core.widgets.decor

Decorative widgets.

### Functions

| | |
|---|---|
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *get_app*() | Get the current active (running) Application. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |

### euporie.core.widgets.decor.add_setting

euporie.core.widgets.decor.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.widgets.decor.get_app

euporie.core.widgets.decor.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.decor.to_filter

euporie.core.widgets.decor.**to_filter**(*bool_or_filter: Union[Filter, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

## Classes

| | |
|---|---|
| *Border*(body, border, style, str] =, show_borders) | Draw a border around any container. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *DiBool*([top, right, bottom, left]) | A tuple of four bools with directions. |
| *DropShadow*([amount]) | A transparent container which makes the background darker. |
| *DynamicContainer*(get_container) | Container class that dynamically returns any Container. |
| *Float*(content[, top, right, bottom, left, ...]) | Float for use in a *FloatContainer*. |
| *FloatContainer*(content, floats[, modal, ...]) | A *FloatContainer* which uses :py`BoundedWritePosition`s. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *Shadow*(body) | Draw a shadow underneath/behind this container. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |

### euporie.core.widgets.decor.Border

**class** euporie.core.widgets.decor.**Border**(*body: AnyContainer*, *border: GridStyle | None = □□□□ □□ □□ □□□□ □□□□*, *style: str | Callable[[], str] = 'class:border'*, *show_borders: DiBool | None = None*)

Draw a border around any container.

### euporie.core.widgets.decor.ConditionalContainer

**class** euporie.core.widgets.decor.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.decor.DiBool

**class** euporie.core.widgets.decor.**DiBool**(*top: bool = False*, *right: bool = False*, *bottom: bool = False*, *left: bool = False*)

A tuple of four bools with directions.

### euporie.core.widgets.decor.DropShadow

**class** euporie.core.widgets.decor.**DropShadow**(*amount: float = 0.5*)

A transparent container which makes the background darker.

### euporie.core.widgets.decor.DynamicContainer

**class** euporie.core.widgets.decor.**DynamicContainer**(*get_container: Callable[[], AnyContainer]*)

Container class that dynamically returns any Container.

**Parameters**

**get_container** – Callable that returns a *Container* instance or any widget with a __pt_container__ method.

### euporie.core.widgets.decor.Float

**class** euporie.core.widgets.decor.**Float**(*content: AnyContainer*, *top: int | None = None*, *right: int | None = None*, *bottom: int | None = None*, *left: int | None = None*, *width: int | Callable[[], int] | None = None*, *height: int | Callable[[], int] | None = None*, *xcursor: bool = False*, *ycursor: bool = False*, *attach_to_window: AnyContainer | None = None*, *hide_when_covering_content: bool = False*, *allow_cover_cursor: bool = False*, *z_index: int = 1*, *transparent: bool = False*)

Float for use in a *FloatContainer*. Except for the *content* parameter, all other options are optional.

**Parameters**

- **content** – *Container* instance.
- **width** – *Dimension* or callable which returns a *Dimension*.
- **height** – *Dimension* or callable which returns a *Dimension*.
- **left** – Distance to the left edge of the *FloatContainer*.
- **right** – Distance to the right edge of the *FloatContainer*.
- **top** – Distance to the top of the *FloatContainer*.
- **bottom** – Distance to the bottom of the *FloatContainer*.
- **attach_to_window** – Attach to the cursor from this window, instead of the current window.
- **hide_when_covering_content** – Hide the float when it covers content underneath.
- **allow_cover_cursor** – When *False*, make sure to display the float below the cursor. Not on top of the indicated position.
- **z_index** – Z-index position. For a Float, this needs to be at least one. It is relative to the z_index of the parent container.
- **transparent** – *Filter* indicating whether this float needs to be drawn transparently.

### euporie.core.widgets.decor.FloatContainer

**class** euporie.core.widgets.decor.**FloatContainer**(*content: AnyContainer*, *floats: list[*Float*]*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str* | *Callable[[], str] = ''*, *z_index: int* | *None = None*)

A *FloatContainer* which uses :py`BoundedWritePosition`s.

### euporie.core.widgets.decor.HSplit

**class** euporie.core.widgets.decor.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str* | *None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int* | *None = None*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str* | *Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.decor.Shadow

**class** euporie.core.widgets.decor.**Shadow**(*body: AnyContainer*)

Draw a shadow underneath/behind this container.

This is a globally configurable version of the `prompt_toolkit.widows.base.Shadow` class.

### euporie.core.widgets.decor.VSplit

**class** euporie.core.widgets.decor.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str* | *None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int* | *None = None*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str* | *Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.decor.Window

**class** euporie.core.widgets.decor.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

**class** euporie.core.widgets.decor.**Border**(*body: AnyContainer*, *border: GridStyle | None = □□□□ □□ □□ □□□□ □□□□*, *style: str | Callable[[], str] = 'class:border'*, *show_borders: DiBool | None = None*)

Bases: object

Draw a border around any container.

**add_style**(*extra: str*) → Callable[[], str]

Return a function which adds a style string to the border style.

**class** euporie.core.widgets.decor.**Shadow**(*body: AnyContainer*)

Bases: object

Draw a shadow underneath/behind this container.

This is a globally configurable version of the prompt_toolkit.widows.base.Shadow class.

## euporie.core.widgets.dialog

Dialog.

### Functions

| | |
|---|---|
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *align*(ft[, how, width, style, placeholder, ...]) | Align formatted text at a given width. |
| *focus_next*(event) | Focus the next visible Window. |
| *focus_previous*(event) | Focus the previous visible Window. |
| *get_app*() | Get the current active (running) Application. |
| *has_focus*(value) | Enable when this buffer has the focus. |
| *lex*(ft, lexer_name) | Format formatted text using a named `pygments` lexer. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *split_lines*(fragments) | Take a single list of (style_str, text) tuples and yield one such list for each line. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |

## euporie.core.widgets.dialog.abstractmethod

euporie.core.widgets.dialog.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):

…

## euporie.core.widgets.dialog.add_cmd

euporie.core.widgets.dialog.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.widgets.dialog.align

euporie.core.widgets.dialog.**align**(*ft: StyleAndTextTuples*, *how:* FormattedTextAlign = *FormattedTextAlign.LEFT*, *width: int | None = None*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Align formatted text at a given width.

**Parameters**

- **how** – The alignment direction
- **ft** – The formatted text to strip
- **width** – The width to which the output should be padded. If `None`, the length of the longest line is used
- **style** – The style to apply to the padding
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – If True, whitespace will be ignored

**Returns**

The aligned formatted text

### euporie.core.widgets.dialog.focus_next

euporie.core.widgets.dialog.**focus_next**(*event:* KeyPressEvent) → None

Focus the next visible Window. (Often bound to the *Tab* key.)

### euporie.core.widgets.dialog.focus_previous

euporie.core.widgets.dialog.**focus_previous**(*event:* KeyPressEvent) → None

Focus the previous visible Window. (Often bound to the *BackTab* key.)

### euporie.core.widgets.dialog.get_app

euporie.core.widgets.dialog.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.dialog.has_focus

euporie.core.widgets.dialog.**has_focus**(*value: FocusableElement*) → *Condition*

Enable when this buffer has the focus.

### euporie.core.widgets.dialog.lex

euporie.core.widgets.dialog.**lex**(*ft: StyleAndTextTuples*, *lexer_name: str*) → StyleAndTextTuples

Format formatted text using a named `pygments` lexer.

### euporie.core.widgets.dialog.register_bindings

euporie.core.widgets.dialog.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.widgets.dialog.split_lines

euporie.core.widgets.dialog.**split_lines**(*fragments: Iterable[OneStyleAndTextTuple]*) → Iterable[StyleAndTextTuples]

Take a single list of (style_str, text) tuples and yield one such list for each line. Just like str.split, this will yield at least one item.

**Parameters**

**fragments** – Iterable of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.dialog.to_formatted_text

euporie.core.widgets.dialog.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

## Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *AboutDialog*(app) | A dialog which shows an "about" message. |
| *Border*(body, border, style, str] =, show_borders) | Draw a border around any container. |
| *Box*(body[, padding, padding_left, ...]) | Add padding around a container. |
| *Button*(text, on_click, None] \| None = None, ...) | A clickable button widget. |
| *ClipboardData*([text, type]) | Text on the clipboard. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *ConfirmDialog*(app) | A dialog which allows the user to confirm an action. |

continues on next page

Table 8 – continued from previous page

| | |
|---|---|
| Dialog(app) | A modal dialog which is displayed above the application. |
| DialogTitleControl(title, dialog, window) | A draggable dialog titlebar. |
| Dimension([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| DynamicContainer(get_container) | Container class that dynamically returns any Container. |
| DynamicKeyBindings(get_key_bindings) | KeyBindings class that can dynamically returns any KeyBindings. |
| ErrorDialog(app) | A dialog to show unhandled exceptions. |
| FileBrowser([path, on_select, on_open, ...]) | A file browser. |
| FileDialog(app) | A base dialog to prompt the user for a file path. |
| Float(content[, top, right, bottom, left, ...]) | Float for use in a FloatContainer. |
| FocusedStyle(body[, style_focus, style_hover]) | Apply a style to child containers when focused or hovered. |
| FormattedTextAlign(value[, names, module, ...]) | Alignment of formatted text. |
| FormattedTextControl([text, style, ...]) | Control that displays formatted text. |
| HSplit(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| KeyBindings() | A container for a set of key bindings. |
| Label(text[, style, width, ...]) | Widget that displays the given text. |
| LabelledWidget(body, label[, style, ...]) | A widget which applies a label to another widget. |
| MouseButton(value[, names, module, ...]) | |
| MouseEventType(value[, names, module, ...]) | |
| MsgBoxDialog(app) | A dialog which shows the user a message. |
| NoKernelsDialog(app) | Dialog to warn the user that no installed kernels were found. |
| OpenFileDialog(app) | A dialog which prompts the user for a filepath to open. |
| Path(*args, **kwargs) | PurePath subclass that can make system calls. |
| PathCompleter([only_directories, get_paths, ...]) | Complete for Path variables. |
| SaveAsDialog(app) | A dialog which prompts the user for a filepath to save the current tab. |
| Select(options, labels, index, indices, ...) | A select widget, which allows one or more items to be selected from a list. |
| SelectKernelDialog(app) | A dialog which allows the user to select a kernel. |
| Shadow(body) | Draw a shadow underneath/behind this container. |
| ShortcutsDialog(app) | Display details of registered key-bindings in a dialog. |
| SimpleCache([maxsize]) | Very simple cache that discards the oldest item when the cache size is exceeded. |
| Tab(app[, path]) | Base class for interface tabs. |
| Text([text, style, height, min_height, ...]) | A text input widget. |
| UIContent(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| UIControl() | Base class for all user interface controls. |
| UnsavedDialog(app) | A dialog prompting the user to save unsaved changes. |
| VSplit(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| Window([content, width, height, z_index, ...]) | Container that holds a control. |
| WritePosition(xpos, ypos, width, height) | |
| partial | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.widgets.dialog.ABCMeta

**class** euporie.core.widgets.dialog.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.widgets.dialog.AboutDialog

**class** euporie.core.widgets.dialog.**AboutDialog**(*app:* BaseApp)

A dialog which shows an "about" message.

### euporie.core.widgets.dialog.Border

**class** euporie.core.widgets.dialog.**Border**(*body: AnyContainer*, *border: GridStyle | None = □□□□ □□ □□ □□□□ □□□□*, *style: str | Callable[[], str] = 'class:border'*, *show_borders: DiBool | None = None*)

Draw a border around any container.

### euporie.core.widgets.dialog.Box

**class** euporie.core.widgets.dialog.**Box**(*body: AnyContainer*, *padding: AnyDimension = None*, *padding_left: AnyDimension = None*, *padding_right: AnyDimension = None*, *padding_top: AnyDimension = None*, *padding_bottom: AnyDimension = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *char: None | str | Callable[[], str] = None*, *modal: bool = False*, *key_bindings:* KeyBindings *| None = None*)

Add padding around a container.

This also makes sure that the parent can provide more space than required by the child. This is very useful when wrapping a small element with a fixed size into a `VSplit` or `HSplit` object. The `HSplit` and `VSplit` try to make sure to adapt respectively the width and height, possibly shrinking other elements. Wrapping something in a `Box` makes it flexible.

**Parameters**

- **body** – Another container object.
- **padding** – The margin to be used around the body. This can be
- **padding_left** (*overridden by*) – *padding_bottom*.
- **padding_right`** – *padding_bottom*.
- **and** (*padding_top*) – *padding_bottom*.
- **style** – A style string.
- **char** – Character to be used for filling the space around the body. (This is supposed to be a character with a terminal width of 1.)

### euporie.core.widgets.dialog.Button

**class** euporie.core.widgets.dialog.**Button**(*text: AnyFormattedText, on_click: Callable[[Button], None] | None = None, on_mouse_down: Callable[[Button], None] | None = None, disabled: FilterOrBool = False, width: int | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = ▯▯ ▯ ▯▯ ▯▯▯▯ ▯▯, show_borders: DiBool | None = None, selected: bool = False, key_bindings: KeyBindingsBase | None = None, mouse_handler: Callable[[MouseEvent], NotImplementedOrNone] | None = None*)

A clickable button widget.

### euporie.core.widgets.dialog.ClipboardData

**class** euporie.core.widgets.dialog.**ClipboardData**(*text: str = '', type:* SelectionType = *SelectionType.CHARACTERS*)

Text on the clipboard.

**Parameters**

- **text** – string
- **type** – *SelectionType*

### euporie.core.widgets.dialog.Condition

**class** euporie.core.widgets.dialog.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.dialog.ConditionalContainer

**class** euporie.core.widgets.dialog.**ConditionalContainer**(*content: AnyContainer, filter:* FilterOrBool)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.dialog.ConfirmDialog

**class** euporie.core.widgets.dialog.**ConfirmDialog**(*app:* BaseApp)

A dialog which allows the user to confirm an action.

### euporie.core.widgets.dialog.Dialog

**class** euporie.core.widgets.dialog.**Dialog**(*app:* BaseApp)

A modal dialog which is displayed above the application.

Returns focus to the previously selected control when closed.

### euporie.core.widgets.dialog.DialogTitleControl

**class** euporie.core.widgets.dialog.**DialogTitleControl**(*title: AnyFormattedText*, *dialog:* Dialog, *window:* Window)

A draggable dialog titlebar.

### euporie.core.widgets.dialog.Dimension

**class** euporie.core.widgets.dialog.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.widgets.dialog.DynamicContainer

**class** euporie.core.widgets.dialog.**DynamicContainer**(*get_container: Callable[[], AnyContainer]*)

Container class that dynamically returns any Container.

**Parameters**

**get_container** – Callable that returns a *Container* instance or any widget with a `__pt_container__` method.

### euporie.core.widgets.dialog.DynamicKeyBindings

**class** euporie.core.widgets.dialog.**DynamicKeyBindings**(*get_key_bindings: Callable[[], prompt_toolkit.key_binding.key_bindings.KeyBindingsBase | None]*)

KeyBindings class that can dynamically returns any KeyBindings.

**Parameters**

**get_key_bindings** – Callable that returns a *KeyBindings* instance.

### euporie.core.widgets.dialog.ErrorDialog

**class** euporie.core.widgets.dialog.**ErrorDialog**(*app:* BaseApp)

A dialog to show unhandled exceptions.

### euporie.core.widgets.dialog.FileBrowser

**class** euporie.core.widgets.dialog.**FileBrowser**(*path: Path | None = None, on_select: Callable[[Path], None] | None = None, on_open: Callable[[Path], None] | None = None, on_chdir: Callable[[Path], None] | None = None, width: AnyDimension = None, height: AnyDimension = None, style: str = '', show_address_bar: FilterOrBool = True*)

A file browser.

### euporie.core.widgets.dialog.FileDialog

**class** euporie.core.widgets.dialog.**FileDialog**(*app:* BaseApp)

A base dialog to prompt the user for a file path.

### euporie.core.widgets.dialog.Float

**class** euporie.core.widgets.dialog.**Float**(*content: AnyContainer, top: int | None = None, right: int | None = None, bottom: int | None = None, left: int | None = None, width: int | Callable[[], int] | None = None, height: int | Callable[[], int] | None = None, xcursor: bool = False, ycursor: bool = False, attach_to_window: AnyContainer | None = None, hide_when_covering_content: bool = False, allow_cover_cursor: bool = False, z_index: int = 1, transparent: bool = False*)

Float for use in a *FloatContainer*. Except for the *content* parameter, all other options are optional.

**Parameters**

- **content** – *Container* instance.
- **width** – *Dimension* or callable which returns a *Dimension*.
- **height** – *Dimension* or callable which returns a *Dimension*.

- **left** – Distance to the left edge of the *FloatContainer*.
- **right** – Distance to the right edge of the *FloatContainer*.
- **top** – Distance to the top of the *FloatContainer*.
- **bottom** – Distance to the bottom of the *FloatContainer*.
- **attach_to_window** – Attach to the cursor from this window, instead of the current window.
- **hide_when_covering_content** – Hide the float when it covers content underneath.
- **allow_cover_cursor** – When *False*, make sure to display the float below the cursor. Not on top of the indicated position.
- **z_index** – Z-index position. For a Float, this needs to be at least one. It is relative to the z_index of the parent container.
- **transparent** – *Filter* indicating whether this float needs to be drawn transparently.

### euporie.core.widgets.dialog.FocusedStyle

**class** euporie.core.widgets.dialog.**FocusedStyle**(*body: AnyContainer*, *style_focus: str | Callable[[], str] = 'class:focused'*, *style_hover: str | Callable[[], str] = ''*)

Apply a style to child containers when focused or hovered.

### euporie.core.widgets.dialog.FormattedTextAlign

**class** euporie.core.widgets.dialog.**FormattedTextAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of formatted text.

### euporie.core.widgets.dialog.FormattedTextControl

**class** euporie.core.widgets.dialog.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.

- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole Window, pass the style to the Window instead.)
- **key_bindings** – a *KeyBindings* object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.widgets.dialog.HSplit

**class** euporie.core.widgets.dialog.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: VerticalAlign* = *VerticalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.dialog.KeyBindings

**class** euporie.core.widgets.dialog.**KeyBindings**

A container for a set of key bindings.

Example usage:

```
kb = KeyBindings()

@kb.add('c-t')
def _(event):
    print('Control-T pressed')

@kb.add('c-a', 'c-b')
def _(event):
    print('Control-A pressed, followed by Control-B')

@kb.add('c-x', filter=is_searching)
def _(event):
    print('Control-X pressed')  # Works only if we are searching.
```

### euporie.core.widgets.dialog.Label

**class** euporie.core.widgets.dialog.**Label**(*text: AnyFormattedText*, *style: str = ''*, *width: AnyDimension = None*, *dont_extend_height: bool = True*, *dont_extend_width: bool = False*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *wrap_lines: FilterOrBool = True*)

Widget that displays the given text. It is not editable or focusable.

**Parameters**

- **text** – Text to display. Can be multiline. All value types accepted by `prompt_toolkit.layout.FormattedTextControl` are allowed, including a callable.
- **style** – A style string.
- **width** – When given, use this width, rather than calculating it from the text size.
- **dont_extend_width** – When *True*, don't take up more width than preferred, i.e. the length of the longest line of the text, or value of *width* parameter, if given. *True* by default
- **dont_extend_height** – When *True*, don't take up more width than the preferred height, i.e. the number of lines of the text. *False* by default.

### euporie.core.widgets.dialog.LabelledWidget

**class** euporie.core.widgets.dialog.**LabelledWidget**(*body: AnyContainer*, *label: AnyFormattedText*, *style: str = 'class:input'*, *vertical: FilterOrBool = False*, *html: FilterOrBool = False*)

A widget which applies a label to another widget.

### euporie.core.widgets.dialog.MouseButton

**class** euporie.core.widgets.dialog.**MouseButton**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.dialog.MouseEventType

**class** euporie.core.widgets.dialog.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.dialog.MsgBoxDialog

**class** euporie.core.widgets.dialog.**MsgBoxDialog**(*app:* BaseApp)

A dialog which shows the user a message.

### euporie.core.widgets.dialog.NoKernelsDialog

**class** euporie.core.widgets.dialog.**NoKernelsDialog**(*app:* BaseApp)

Dialog to warn the user that no installed kernels were found.

### euporie.core.widgets.dialog.OpenFileDialog

**class** euporie.core.widgets.dialog.**OpenFileDialog**(*app:* BaseApp)

A dialog which prompts the user for a filepath to open.

### euporie.core.widgets.dialog.Path

**class** euporie.core.widgets.dialog.**Path**(**args*, ***kwargs*)

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

### euporie.core.widgets.dialog.PathCompleter

**class** euporie.core.widgets.dialog.**PathCompleter**(*only_directories: bool = False*, *get_paths: Optional[Callable[[], list[str]]] = None*, *file_filter: Optional[Callable[[str], bool]] = None*, *min_input_len: int = 0*, *expanduser: bool = False*)

Complete for Path variables.

**Parameters**

- **get_paths** – Callable which returns a list of directories to look into when the user enters a relative path.
- **file_filter** – Callable which takes a filename and returns whether this file should show up in the completion. `None` when no filtering has to be done.
- **min_input_len** – Don't do autocompletion when the input string is shorter.

### euporie.core.widgets.dialog.SaveAsDialog

**class** euporie.core.widgets.dialog.**SaveAsDialog**(*app:* BaseApp)

A dialog which prompts the user for a filepath to save the current tab.

### euporie.core.widgets.dialog.Select

**class** euporie.core.widgets.dialog.**Select**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input,select', rows: int | None = 3, prefix: tuple[str, str] = ('', ''), border: GridStyle | None = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None, disabled: FilterOrBool = False, dont_extend_width: FilterOrBool = True, dont_extend_height: FilterOrBool = True*)

A select widget, which allows one or more items to be selected from a list.

### euporie.core.widgets.dialog.SelectKernelDialog

**class** euporie.core.widgets.dialog.**SelectKernelDialog**(*app:* BaseApp)

A dialog which allows the user to select a kernel.

### euporie.core.widgets.dialog.Shadow

**class** euporie.core.widgets.dialog.**Shadow**(*body: AnyContainer*)

Draw a shadow underneath/behind this container.

This is a globally configurable version of the `prompt_toolkit.widows.base.Shadow` class.

### euporie.core.widgets.dialog.ShortcutsDialog

**class** euporie.core.widgets.dialog.**ShortcutsDialog**(*app:* BaseApp)

Display details of registered key-bindings in a dialog.

### euporie.core.widgets.dialog.SimpleCache

**class** euporie.core.widgets.dialog.**SimpleCache**(*maxsize: int = 8*)

Very simple cache that discards the oldest item when the cache size is exceeded.

**Parameters**

**maxsize** – Maximum size of the cache. (Don't make it too big.)

### euporie.core.widgets.dialog.Tab

**class** euporie.core.widgets.dialog.**Tab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Base class for interface tabs.

### euporie.core.widgets.dialog.Text

**class** euporie.core.widgets.dialog.**Text**(*text: str = ''*, *style: str = 'class:input'*, *height: int = 1*, *min_height: int = 1*, *multiline: FilterOrBool = False*, *expand: FilterOrBool = True*, *width: int | None = None*, *completer: Completer | None = None*, *options: list[str] | Callable[[], list[str]] | None = None*, *show_borders: DiBool | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *validation: Callable[[str], bool] | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *placeholder: str | None = None*, *lexer: Lexer | None = None*, *input_processors: Sequence[Processor] | None = None*, *disabled: FilterOrBool = False*, *password: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *prompt: AnyFormattedText | None = None*)

A text input widget.

### euporie.core.widgets.dialog.UIContent

**class** euporie.core.widgets.dialog.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>*, *line_count: int = 0*, *cursor_position: Point | None = None*, *menu_position: Point | None = None*, *show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.widgets.dialog.UIControl

**class** euporie.core.widgets.dialog.**UIControl**

Base class for all user interface controls.

### euporie.core.widgets.dialog.UnsavedDialog

**class** euporie.core.widgets.dialog.**UnsavedDialog**(*app:* BaseApp)

A dialog prompting the user to save unsaved changes.

### euporie.core.widgets.dialog.VSplit

**class** euporie.core.widgets.dialog.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: HorizontalAlign* = *HorizontalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.dialog.Window

**class** euporie.core.widgets.dialog.**Window**(*content:* UIControl | *None* = *None*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *dont_extend_width: FilterOrBool* = *False*, *dont_extend_height: FilterOrBool* = *False*, *ignore_content_width: FilterOrBool* = *False*, *ignore_content_height: FilterOrBool* = *False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool* = *False*, *wrap_lines: FilterOrBool* = *False*, *get_vertical_scroll: Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool* = *False*, *cursorline: FilterOrBool* = *False*, *cursorcolumn: FilterOrBool* = *False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]]* = *None*, *align:* WindowAlign | *Callable[[],* WindowAlign*]* = *WindowAlign.LEFT*, *style: str* | *Callable[[], str]* = *''*, *char: None* | *str* | *Callable[[], str]* = *None*, *get_line_prefix: GetLinePrefixCallable* | *None* = *None*)

Container that holds a control.

### euporie.core.widgets.dialog.WritePosition

**class** euporie.core.widgets.dialog.**WritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*)

### euporie.core.widgets.dialog.partial

**class** euporie.core.widgets.dialog.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.widgets.dialog.**AboutDialog**(*app:* BaseApp)

Bases: *Dialog*

A dialog which shows an "about" message.

**body_padding_bottom = 0**

**body_padding_top = 1**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None

Hide the dialog.

**load**() → None

Load the dialog's body.

**show**(**params: Any*) → None

Display and focuses the dialog.

**title: AnyFormattedText | None = 'About'**

**toggle**() → None

Show or hides the dialog.

**class** euporie.core.widgets.dialog.**ConfirmDialog**(*app:* BaseApp)

Bases: *Dialog*

A dialog which allows the user to confirm an action.

**body_padding_bottom = 0**

**body_padding_top = 1**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None

Hide the dialog.

**load**(*title: str = 'Are you sure?'*, *message: str = 'Please confirm'*, *cb: Callable[[], None] | None = None*) → None

Load dialog body & buttons.

**show**(**params: Any*) → None

Display and focuses the dialog.

**title: AnyFormattedText | None = None**

**toggle**() → None

Show or hides the dialog.

**class** euporie.core.widgets.dialog.**Dialog**(*app:* BaseApp)

Bases: *Float*

A modal dialog which is displayed above the application.

Returns focus to the previously selected control when closed.

**body_padding_bottom = 0**

**body_padding_top = 1**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None

Hide the dialog.

**abstract load**() → None

Load the dialog's body etc.

**show**(***params: Any*) → None

Display and focuses the dialog.

**title: AnyFormattedText | None = None**

**toggle**() → None

Show or hides the dialog.

**class** euporie.core.widgets.dialog.**DialogTitleControl**(*title: AnyFormattedText*, *dialog:* Dialog, *window:* Window)

Bases: *UIControl*

A draggable dialog titlebar.

**create_content**(*width: int*, *height: int* | *None*) → *UIContent*

Create the title text 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**() → *KeyBindingsBase* | None

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

Move the dialog when the titlebar is dragged.

**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: GetLinePrefixCallable | None*) → int | None

**preferred_width**(*max_available_width: int*) → int | None

**reset**() → None

**class** euporie.core.widgets.dialog.**ErrorDialog**(*app:* BaseApp)

Bases: *Dialog*

A dialog to show unhandled exceptions.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 1**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None

Hide the dialog.

**last_focused: FocusableElement | None**

**load**(*exception: Exception | None* = *None*, *when: str* = *''*) → None

Load dialog body & buttons.

**show**(***params: Any*) → None

Display and focuses the dialog.

**title: AnyFormattedText | None = 'Error'**

**to_focus: FocusableElement | None**

**toggle**() → None

Show or hides the dialog.

**class** euporie.core.widgets.dialog.**FileDialog**(*app:* BaseApp)

Bases: *Dialog*

A base dialog to prompt the user for a file path.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 1**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None

Hide the dialog.

**last_focused: FocusableElement | None**

**load**(*text: str* = '', *tab:* Tab | *None* = *None*, *error: str* = '', *cb: Callable* | *None* = *None*) → None

Load the dialog body.

**show**(***params: Any*) → None

Display and focuses the dialog.

**title: AnyFormattedText | None = 'Select a File'**

**to_focus: FocusableElement | None**

**toggle**() → None

Show or hides the dialog.

**validate**(*buffer:* Buffer, *tab:* Tab | *None*, *cb: Callable* | *None* = *None*) → None

Validate the input.

**class** euporie.core.widgets.dialog.**MsgBoxDialog**(*app:* BaseApp)

Bases: *Dialog*

A dialog which shows the user a message.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 1**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None

Hide the dialog.

**last_focused: FocusableElement | None**

**load**(*title: str* = 'Message', *message: str* = '') → None

Load dialog body & buttons.

**show**(***params: Any*) → None

Display and focuses the dialog.

**title: AnyFormattedText | None = 'Message'**

**to_focus: FocusableElement | None**

**toggle**() → None
Show or hides the dialog.

**class** euporie.core.widgets.dialog.**NoKernelsDialog**(*app:* BaseApp)
Bases: *Dialog*

Dialog to warn the user that no installed kernels were found.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 1**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None
Hide the dialog.

**last_focused: FocusableElement | None**

**load**() → None
Load the dialog body.

**show**(**params: Any*) → None
Display and focuses the dialog.

**title: AnyFormattedText | None = 'No Kernels Found'**

**to_focus: FocusableElement | None**

**toggle**() → None
Show or hides the dialog.

**class** euporie.core.widgets.dialog.**OpenFileDialog**(*app:* BaseApp)
Bases: *FileDialog*

A dialog which prompts the user for a filepath to open.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 1**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None
Hide the dialog.

**last_focused: FocusableElement | None**

**load**(*text: str = ''*, *tab: Tab | None = None*, *error: str = ''*, *cb: Callable | None = None*) → None
Load the dialog body.

**show**(***params: Any*) → None
Display and focuses the dialog.

**title: AnyFormattedText | None = 'Select a File to Open'**

**to_focus: FocusableElement | None**

**toggle**() → None
Show or hides the dialog.

**validate**(*buffer: Buffer*, *tab: Tab | None*, *cb: Callable | None = None*) → None
Validate the the file to open exists.

**class** euporie.core.widgets.dialog.**SaveAsDialog**(*app: BaseApp*)
Bases: *FileDialog*

A dialog which prompts the user for a filepath to save the current tab.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 1**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event: KeyPressEvent | None = None*) → None
Hide the dialog.

**last_focused: FocusableElement | None**

**load**(*text: str = ''*, *tab: Tab | None = None*, *error: str = ''*, *cb: Callable | None = None*) → None
Load the dialog body.

**show**(***params: Any*) → None
Display and focuses the dialog.

**title: AnyFormattedText | None = 'Select a Path to Save'**

**to_focus: FocusableElement | None**

**toggle**() → None
Show or hides the dialog.

**validate**(*buffer: Buffer*, *tab: Tab | None*, *cb: Callable | None = None*) → None
Validate the the file to open exists.

**class** euporie.core.widgets.dialog.**SelectKernelDialog**(*app:* BaseApp)

Bases: *Dialog*

A dialog which allows the user to select a kernel.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 1**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None

Hide the dialog.

**last_focused: FocusableElement | None**

**load**(*kernel_specs: dict[str, Any]* | *None* = *None*, *runtime_dirs: dict[str,* Path*]* | *None* = *None*, *tab:* KernelTab | *None* = *None*, *message: str* = '') → None

Load dialog body & buttons.

**show**(***params: Any*) → None

Display and focuses the dialog.

**title: AnyFormattedText | None = 'Select Kernel'**

**to_focus: FocusableElement | None**

**toggle**() → None

Show or hides the dialog.

**class** euporie.core.widgets.dialog.**ShortcutsDialog**(*app:* BaseApp)

Bases: *Dialog*

Display details of registered key-bindings in a dialog.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 1**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**format_key_info**() → StyleAndTextTuples

Generate a table with the current key bindings.

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None
Hide the dialog.

**last_focused: FocusableElement | None**

**load**(**args: Any*, ***kwargs: Any*) → None
Load the dialog body.

**show**(***params: Any*) → None
Display and focuses the dialog.

**title: AnyFormattedText | None = 'Keyboard Shortcuts'**

**to_focus: FocusableElement | None**

**toggle**() → None
Show or hides the dialog.

**class** euporie.core.widgets.dialog.**UnsavedDialog**(*app:* BaseApp)
Bases: *Dialog*

A dialog prompting the user to save unsaved changes.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 1**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None
Hide the dialog.

**last_focused: FocusableElement | None**

**load**(*tab:* Tab | *None* = *None*, *cb: Callable[[], None]* | *None* = *None*) → None
Load the dialog body.

**show**(***params: Any*) → None
Display and focuses the dialog.

**title: AnyFormattedText | None = 'Unsaved Changes'**

**to_focus: FocusableElement | None**

**toggle**() → None
Show or hides the dialog.

### euporie.core.widgets.display

Define custom controls which re-render on resize.

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *cast*(typ, val) | Cast a value to a type. |
| *ceil*(x, /) | Return the ceiling of x as an Integral. |
| *fragment_list_width*(fragments) | Return the character width of this text fragment list. |
| *get_app*() | Get the current active (running) Application. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *run_in_thread_with_context*(func, *args[, daemon]) | Run a function in an thread, but make sure it uses the same contextvars. |
| *scrollable*(window) | Return a filter which indicates if a window is scrollable. |
| *split_lines*(fragments) | Take a single list of (style_str, text) tuples and yield one such list for each line. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |
| *to_str*(value) | Turn callable or string into string. |
| *wrap*(ft, width[, style, placeholder, left, ...]) | Wrap formatted text at a given width. |

### euporie.core.widgets.display.add_cmd

euporie.core.widgets.display.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.widgets.display.cast

euporie.core.widgets.display.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.widgets.display.ceil

euporie.core.widgets.display.**ceil**(*x*, */*)

Return the ceiling of x as an Integral.

This is the smallest integer >= x.

### euporie.core.widgets.display.fragment_list_width

euporie.core.widgets.display.**fragment_list_width**(*fragments: StyleAndTextTuples*) → int

Return the character width of this text fragment list. (Take double width characters into account.)

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.display.get_app

euporie.core.widgets.display.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.display.load_registered_bindings

euporie.core.widgets.display.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.core.widgets.display.register_bindings

euporie.core.widgets.display.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.widgets.display.run_in_thread_with_context

euporie.core.widgets.display.**run_in_thread_with_context**(*func: Callable*, **args: Any*, *daemon: bool = True*, ***kwargs: Any*) → None

Run a function in an thread, but make sure it uses the same contextvars.

This is required so that the function will see the right application.

### euporie.core.widgets.display.scrollable

euporie.core.widgets.display.**scrollable**(*window:* Window) → *Filter*

Return a filter which indicates if a window is scrollable.

### euporie.core.widgets.display.split_lines

euporie.core.widgets.display.**split_lines**(*fragments: Iterable[OneStyleAndTextTuple]*) → Iterable[StyleAndTextTuples]

Take a single list of (style_str, text) tuples and yield one such list for each line. Just like str.split, this will yield at least one item.

**Parameters**

**fragments** – Iterable of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.display.to_filter

euporie.core.widgets.display.**to_filter**(*bool_or_filter: Union[*Filter, *bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### euporie.core.widgets.display.to_str

euporie.core.widgets.display.**to_str**(*value: Union[Callable[[], str], str]*) → str

Turn callable or string into string.

### euporie.core.widgets.display.wrap

euporie.core.widgets.display.**wrap**(*ft: StyleAndTextTuples*, *width: int*, *style: str = ''*, *placeholder: str = '…'*, *left: int = 0*, *truncate_long_words: bool = True*, *strip_trailing_ws: bool = False*, *margin: str = ''*) → StyleAndTextTuples

Wrap formatted text at a given width.

If words are longer than the given line they will be truncated

**Parameters**

- **ft** – The formatted text to wrap
- **width** – The width at which to wrap the text
- **style** – The style to apply to the truncation placeholder
- **placeholder** – The string that will appear at the end of a truncated line
- **left** – The starting position within the first line
- **truncate_long_words** – If `True` words longer than a line will be truncated
- **strip_trailing_ws** – If `True`, trailing whitespace will be removed from the ends of lines
- **margin** – Text to use a margin for the continuation of wrapped lines

**Returns**

The wrapped formatted text

## Classes

| | |
|---|---|
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *Datum*(data, *args, **kwargs) | Class for storing and converting display data. |
| *Display*(datum[, height, width, focusable, ...]) | Rich output displays. |
| *DisplayControl*(datum[, focusable, ...]) | Web view displays. |
| *DisplayWindow*([content, width, height, ...]) | A window sub-class which can scroll left and right. |
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *FastDictCache*(get_value[, size]) | Fast, lightweight cache which keeps at most *size* items. |
| *GraphicProcessor*(control) | Class which loads and positions graphics references in a *UIContent*. |
| *MarginContainer*(margin, target) | A container which renders a stand-alone margin. |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, sent to *UIControl.mouse_handler*. |
| *MouseEventType*(value[, names, module, ...]) | |
| *Point*(x, y) | |
| *ScrollbarMargin*([display_arrows, ...]) | Margin displaying a scrollbar. |
| *SimpleCache*([maxsize]) | Very simple cache that discards the oldest item when the cache size is exceeded. |
| *Size*(rows, columns) | |
| *UIContent*(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| *UIControl*() | Base class for all user interface controls. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.widgets.display.ConditionalContainer

**class** euporie.core.widgets.display.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.display.Datum

**class** euporie.core.widgets.display.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Class for storing and converting display data.

### euporie.core.widgets.display.Display

**class** euporie.core.widgets.display.**Display**(*datum:* Datum, *height: AnyDimension = None*, *width: AnyDimension = None*, *focusable: FilterOrBool = False*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *always_hide_cursor: FilterOrBool = True*, *scrollbar: FilterOrBool = True*, *scrollbar_autohide: FilterOrBool = True*, *dont_extend_height: FilterOrBool = True*, *dont_extend_width: FilterOrBool = False*, *style: str | Callable[[], str] = ''*)

Rich output displays.

A container for displaying rich output data.

### euporie.core.widgets.display.DisplayControl

**class** euporie.core.widgets.display.**DisplayControl**(*datum:* Datum, *focusable: FilterOrBool = False*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *threaded: bool = False*)

Web view displays.

A control which displays rendered HTML content.

### euporie.core.widgets.display.DisplayWindow

**class** euporie.core.widgets.display.**DisplayWindow**(*content: UIControl | None = None, width: AnyDimension = None, height: AnyDimension = None, z_index: int | None = None, dont_extend_width: FilterOrBool = False, dont_extend_height: FilterOrBool = False, ignore_content_width: FilterOrBool = False, ignore_content_height: FilterOrBool = False, left_margins: Sequence[Margin] | None = None, right_margins: Sequence[Margin] | None = None, scroll_offsets: ScrollOffsets | None = None, allow_scroll_beyond_bottom: FilterOrBool = False, wrap_lines: FilterOrBool = False, get_vertical_scroll: Callable[[Window], int] | None = None, get_horizontal_scroll: Callable[[Window], int] | None = None, always_hide_cursor: FilterOrBool = False, cursorline: FilterOrBool = False, cursorcolumn: FilterOrBool = False, colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None, align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT, style: str | Callable[[], str] = '', char: None | str | Callable[[], str] = None, get_line_prefix: GetLinePrefixCallable | None = None*)

A window sub-class which can scroll left and right.

### euporie.core.widgets.display.Event

**class** euporie.core.widgets.display.**Event**(*sender: _Sender, handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.widgets.display.FastDictCache

**class** euporie.core.widgets.display.**FastDictCache**(*get_value: Callable[[...], _V]*, *size: int = 1000000*)

Fast, lightweight cache which keeps at most *size* items. It will discard the oldest items in the cache first.

The cache is a dictionary, which doesn't keep track of access counts. It is perfect to cache little immutable objects which are not expensive to create, but where a dictionary lookup is still much faster than an object instantiation.

**Parameters**

**get_value** – Callable that's called in case of a missing key.

### euporie.core.widgets.display.GraphicProcessor

**class** euporie.core.widgets.display.**GraphicProcessor**(*control:* UIControl)

Class which loads and positions graphics references in a *UIContent*.

### euporie.core.widgets.display.MarginContainer

**class** euporie.core.widgets.display.**MarginContainer**(*margin:* Margin, *target: ScrollableContainer*)

A container which renders a stand-alone margin.

### euporie.core.widgets.display.MouseEvent

**class** euporie.core.widgets.display.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers: frozenset[*prompt_toolkit.mouse_events.MouseModifier*]*)

Mouse event, sent to *UIControl.mouse_handler*.

**Parameters**

- **position** – *Point* instance.
- **event_type** – *MouseEventType*.

### euporie.core.widgets.display.MouseEventType

**class** euporie.core.widgets.display.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.display.Point

**class** euporie.core.widgets.display.**Point**(*x*, *y*)

### euporie.core.widgets.display.ScrollbarMargin

**class** euporie.core.widgets.display.**ScrollbarMargin**(*display_arrows: Union[*Filter*, bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[*Filter*, bool] = False*, *smooth: bool = True*, *style: str = ''*)

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

### euporie.core.widgets.display.SimpleCache

**class** euporie.core.widgets.display.**SimpleCache**(*maxsize: int = 8*)

Very simple cache that discards the oldest item when the cache size is exceeded.

**Parameters**

**maxsize** – Maximum size of the cache. (Don't make it too big.)

### euporie.core.widgets.display.Size

**class** euporie.core.widgets.display.**Size**(*rows*, *columns*)

### euporie.core.widgets.display.UIContent

**class** euporie.core.widgets.display.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>*, *line_count: int = 0*, *cursor_position: Point | None = None*, *menu_position: Point | None = None*, *show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.

- **show_cursor** – Make the cursor visible.

### euporie.core.widgets.display.UIControl

**class** euporie.core.widgets.display.**UIControl**

Base class for all user interface controls.

### euporie.core.widgets.display.VSplit

**class** euporie.core.widgets.display.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.display.Window

**class** euporie.core.widgets.display.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.core.widgets.display.partial

**class** euporie.core.widgets.display.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.widgets.display.**Display**(*datum:* Datum, *height: AnyDimension = None*, *width: AnyDimension = None*, *focusable: FilterOrBool = False*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *always_hide_cursor: FilterOrBool = True*, *scrollbar: FilterOrBool = True*, *scrollbar_autohide: FilterOrBool = True*, *dont_extend_height: FilterOrBool = True*, *dont_extend_width: FilterOrBool = False*, *style: str | Callable[[], str] = ''*)

Bases: object

Rich output displays.

A container for displaying rich output data.

**property datum: Any**

Return the display's current data.

**style**() → str

Use the background color of the data as the default style.

**class** euporie.core.widgets.display.**DisplayControl**(*datum:* Datum, *focusable: FilterOrBool = False*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *threaded: bool = False*)

Bases: *UIControl*

Web view displays.

A control which displays rendered HTML content.

**property content_width: int**

Return the width of the content.

**create_content**(*width: int*, *height: int*) → *UIContent*

Generate the content for this user control.

**Returns**

A *UIContent* instance.

**property cursor_position: *Point***

Get the cursor position.

**property datum: Any**

Return the control's display data.

**get_content**(*datum:* Datum, *width: int*, *height: int*, *loading: bool*, *cursor_position:* Point, *color_palette:* ColorPalette) → *UIContent*

Create a cacheable UIContent.

**get_invalidate_events**() → Iterable[*Event*[object]]

Return the Window invalidate events.

**get_key_bindings**() → *KeyBindingsBase* | None

Return key bindings that are specific for this user control.

**Returns**

**A *KeyBindings* object if some key bindings are specified, or** *None* otherwise.

**get_lines**(*datum:* Datum, *width: int* | *None*, *height: int* | *None*, *fg: str*, *bg: str*, *wrap_lines: bool* = *False*) → list[StyleAndTextTuples]

Render the lines to display in the control.

**get_max_line_width**(*datum:* Datum, *width: int* | *None*, *height: int* | *None*, *wrap_lines: bool* = *False*) → int

Get the maximum lines width for a given rendering.

**is_focusable**() → bool

Tell whether this user control is focusable.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Mouse handler for this control.

**move_cursor_down**() → None

Move the cursor down one line.

**move_cursor_left**() → None

Move the cursor down one line.

**move_cursor_right**() → None

Move the cursor up one line.

**move_cursor_up**() → None

Move the cursor up one line.

**preferred_height**(*width: int*, *max_available_height: int*, *wrap_lines: bool*, *get_line_prefix:* *GetLinePrefixCallable* | *None*) → int | None

Calculate and return the preferred height of the control.

**preferred_width**(*max_available_width: int*) → int | None

Calculate and return the preferred width of the control.

**render**() → None

Render the HTML DOM in a thread.

**reset**() → None

Reset the state of the control.

**class** euporie.core.widgets.display.**DisplayWindow**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Bases: *Window*

A window sub-class which can scroll left and right.

**content: *DisplayControl***

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

Return the list of child *Container* objects.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Calculate the preferred height for this window.

**preferred_width**(*max_available_width: int*) → *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…)

**vertical_scroll: int**

**write_to_screen**(*screen: Screen*, *mouse_handlers: MouseHandlers*, *write_position: WritePosition*, *parent_style: str*, *erase_bg: bool*, *z_index: int | None*) → None

Write window to screen.

### euporie.core.widgets.file_browser

Define a file browser widget.

### Functions

| | |
|---|---|
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *get_app*() | Get the current active (running) Application. |
| *is_dir*(path) | Check if a path is a directory. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |

### euporie.core.widgets.file_browser.add_setting

euporie.core.widgets.file_browser.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.widgets.file_browser.get_app

euporie.core.widgets.file_browser.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.file_browser.is_dir

euporie.core.widgets.file_browser.**is_dir**(*path: str | pathlib.Path*) → bool | None

Check if a path is a directory.

### euporie.core.widgets.file_browser.to_filter

euporie.core.widgets.file_browser.**to_filter**(*bool_or_filter: Union[Filter, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

## Classes

| | |
|---|---|
| Border(body, border, style, str] =, show_borders) | Draw a border around any container. |
| Button(text, on_click, None] \| None = None, ...) | A clickable button widget. |
| ConditionalContainer(content, filter) | Wrapper around any other container that can change the visibility. |
| Event(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| FastDictCache(get_value[, size]) | Fast, lightweight cache which keeps at most *size* items. |
| FileBrowser([path, on_select, on_open, ...]) | A file browser. |
| FileBrowserControl([path, on_chdir, ...]) | A control for browsing a filesystem. |
| FocusedStyle(body[, style_focus, style_hover]) | Apply a style to child containers when focused or hovered. |
| HSplit(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| KeyBindings() | A container for a set of key bindings. |
| KeyBindingsBase() | Interface for a KeyBindings. |
| MarginContainer(margin, target) | A container which renders a stand-alone margin. |
| MouseButton(value[, names, module, ...]) | |
| MouseEvent(position, event_type, button, ...) | Mouse event, sent to *UIControl.mouse_handler*. |
| MouseEventType(value[, names, module, ...]) | |
| Path(*args, **kwargs) | PurePath subclass that can make system calls. |
| PathCompleter([only_directories, get_paths, ...]) | Complete for Path variables. |
| Point(x, y) | |
| ScrollbarMargin([display_arrows, ...]) | Margin displaying a scrollbar. |
| Text([text, style, height, min_height, ...]) | A text input widget. |
| UIContent(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| UIControl() | Base class for all user interface controls. |
| UPath(*args[, protocol]) | |
| VSplit(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| Window([content, width, height, z_index, ...]) | Container that holds a control. |
| WritePosition(xpos, ypos, width, height) | |

### euporie.core.widgets.file_browser.Border

**class** euporie.core.widgets.file_browser.**Border**(*body: AnyContainer*, *border: GridStyle | None = □□□□ □ □□ □□□□ □□□□*, *style: str | Callable[[], str] = 'class:border'*, *show_borders: DiBool | None = None*)

Draw a border around any container.

### euporie.core.widgets.file_browser.Button

**class** euporie.core.widgets.file_browser.**Button**(*text: AnyFormattedText, on_click: Callable[[Button], None] | None = None, on_mouse_down: Callable[[Button], None] | None = None, disabled: FilterOrBool = False, width: int | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = ▯▯ ▯ ▯▯ ▯▯▯▯ ▯▯, show_borders: DiBool | None = None, selected: bool = False, key_bindings: KeyBindingsBase | None = None, mouse_handler: Callable[[MouseEvent], NotImplementedOrNone] | None = None*)

A clickable button widget.

### euporie.core.widgets.file_browser.ConditionalContainer

**class** euporie.core.widgets.file_browser.**ConditionalContainer**(*content: AnyContainer, filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.file_browser.Event

**class** euporie.core.widgets.file_browser.**Event**(*sender: _Sender, handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.widgets.file_browser.FastDictCache

**class** euporie.core.widgets.file_browser.**FastDictCache**(*get_value: Callable[[...], _V]*, *size: int = 1000000*)

Fast, lightweight cache which keeps at most *size* items. It will discard the oldest items in the cache first.

The cache is a dictionary, which doesn't keep track of access counts. It is perfect to cache little immutable objects which are not expensive to create, but where a dictionary lookup is still much faster than an object instantiation.

**Parameters**

**get_value** – Callable that's called in case of a missing key.

### euporie.core.widgets.file_browser.FileBrowser

**class** euporie.core.widgets.file_browser.**FileBrowser**(*path: Path | None = None*, *on_select: Callable[[Path], None] | None = None*, *on_open: Callable[[Path], None] | None = None*, *on_chdir: Callable[[Path], None] | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *show_address_bar: FilterOrBool = True*)

A file browser.

### euporie.core.widgets.file_browser.FileBrowserControl

**class** euporie.core.widgets.file_browser.**FileBrowserControl**(*path: Path | None = None*, *on_chdir: Callable[[FileBrowserControl], None] | None = None*, *on_select: Callable[[FileBrowserControl], None] | None = None*, *on_open: Callable[[FileBrowserControl], None] | None = None*, *window: Window | None = None*)

A control for browsing a filesystem.

### euporie.core.widgets.file_browser.FocusedStyle

**class** euporie.core.widgets.file_browser.**FocusedStyle**(*body: AnyContainer*, *style_focus: str | Callable[[], str] = 'class:focused'*, *style_hover: str | Callable[[], str] = ''*)

Apply a style to child containers when focused or hovered.

### euporie.core.widgets.file_browser.HSplit

**class** euporie.core.widgets.file_browser.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.file_browser.KeyBindings

**class** euporie.core.widgets.file_browser.**KeyBindings**

A container for a set of key bindings.

Example usage:

```
kb = KeyBindings()

@kb.add('c-t')
def _(event):
    print('Control-T pressed')

@kb.add('c-a', 'c-b')
def _(event):
    print('Control-A pressed, followed by Control-B')

@kb.add('c-x', filter=is_searching)
def _(event):
    print('Control-X pressed')  # Works only if we are searching.
```

### euporie.core.widgets.file_browser.KeyBindingsBase

**class** euporie.core.widgets.file_browser.**KeyBindingsBase**

Interface for a KeyBindings.

### euporie.core.widgets.file_browser.MarginContainer

**class** euporie.core.widgets.file_browser.**MarginContainer**(*margin: Margin*, *target: ScrollableContainer*)

A container which renders a stand-alone margin.

### euporie.core.widgets.file_browser.MouseButton

**class** euporie.core.widgets.file_browser.**MouseButton**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.file_browser.MouseEvent

**class** euporie.core.widgets.file_browser.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers:* *frozenset*[prompt_toolkit.mouse_events.MouseModifier])

Mouse event, sent to *UIControl.mouse_handler*.

**Parameters**

- **position** – *Point* instance.
- **event_type** – *MouseEventType*.

### euporie.core.widgets.file_browser.MouseEventType

**class** euporie.core.widgets.file_browser.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.file_browser.Path

**class** euporie.core.widgets.file_browser.**Path**(**args*, ***kwargs*)

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

### euporie.core.widgets.file_browser.PathCompleter

**class** euporie.core.widgets.file_browser.**PathCompleter**(*only_directories: bool = False*, *get_paths: Optional[Callable[[], list[str]]] = None*, *file_filter: Optional[Callable[[str], bool]] = None*, *min_input_len: int = 0*, *expanduser: bool = False*)

Complete for Path variables.

**Parameters**

- **get_paths** – Callable which returns a list of directories to look into when the user enters a relative path.
- **file_filter** – Callable which takes a filename and returns whether this file should show up in the completion. `None` when no filtering has to be done.
- **min_input_len** – Don't do autocompletion when the input string is shorter.

### euporie.core.widgets.file_browser.Point

**class** euporie.core.widgets.file_browser.**Point**(*x*, *y*)

### euporie.core.widgets.file_browser.ScrollbarMargin

**class** euporie.core.widgets.file_browser.**ScrollbarMargin**(*display_arrows: Union[Filter, bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[Filter, bool] = False*, *smooth: bool = True*, *style: str = ''*)

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

### euporie.core.widgets.file_browser.Text

**class** euporie.core.widgets.file_browser.**Text**(*text: str = ''*, *style: str = 'class:input'*, *height: int = 1*, *min_height: int = 1*, *multiline: FilterOrBool = False*, *expand: FilterOrBool = True*, *width: int | None = None*, *completer: Completer | None = None*, *options: list[str] | Callable[[], list[str]] | None = None*, *show_borders: DiBool | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *validation: Callable[[str], bool] | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *placeholder: str | None = None*, *lexer: Lexer | None = None*, *input_processors: Sequence[Processor] | None = None*, *disabled: FilterOrBool = False*, *password: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *prompt: AnyFormattedText | None = None*)

A text input widget.

### euporie.core.widgets.file_browser.UIContent

**class** euporie.core.widgets.file_browser.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>, line_count: int = 0, cursor_position: Point | None = None, menu_position: Point | None = None, show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.widgets.file_browser.UIControl

**class** euporie.core.widgets.file_browser.**UIControl**

Base class for all user interface controls.

### euporie.core.widgets.file_browser.UPath

**class** euporie.core.widgets.file_browser.**UPath**(**args, protocol: str | None = None, **storage_options: Any*)

### euporie.core.widgets.file_browser.VSplit

**class** euporie.core.widgets.file_browser.**VSplit**(*children: Sequence[AnyContainer], window_too_small: Container | None = None, align: HorizontalAlign = HorizontalAlign.JUSTIFY, padding: AnyDimension = 0, padding_char: str | None = None, padding_style: str = '', width: AnyDimension = None, height: AnyDimension = None, z_index: int | None = None, modal: bool = False, key_bindings: KeyBindingsBase | None = None, style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.file_browser.Window

**class** euporie.core.widgets.file_browser.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.core.widgets.file_browser.WritePosition

**class** euporie.core.widgets.file_browser.**WritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*)

**class** euporie.core.widgets.file_browser.**FileBrowser**(*path: Path | None = None*, *on_select: Callable[[Path], None] | None = None*, *on_open: Callable[[Path], None] | None = None*, *on_chdir: Callable[[Path], None] | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *show_address_bar: FilterOrBool = True*)

Bases: object

A file browser.

**completer = <prompt_toolkit.completion.filesystem.PathCompleter object>**

**class** euporie.core.widgets.file_browser.**FileBrowserControl**(*path:* Path | *None* = *None*, *on_chdir: Callable[[*FileBrowserControl*], None]* | *None* = *None*, *on_select: Callable[[*FileBrowserControl*], None]* | *None* = *None*, *on_open: Callable[[*FileBrowserControl*], None]* | *None* = *None*, *window:* Window | *None* = *None*)

Bases: *UIControl*

A control for browsing a filesystem.

**property contents: list[tuple[bool, *pathlib.Path*]]**

Return the contents of the current folder.

**create_content**(*width: int*, *height: int*) → *UIContent*

Generate the content for this user control.

**property dir: *Path***

Return the current folder path.

**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**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

Key bindings specific to this user control.

**hover**(*row: int* | *None*) → NotImplementedOrNone

Hover a file in the browser.

**is_focusable**() → bool

Determine that the file_browser is focusable.

**static load_path**(*path:* Path) → list[tuple[bool, *pathlib.Path*]]

Return the contents of a folder.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

**move_cursor_down**() → None

Request to move the cursor down.

**move_cursor_up**() → None

Request to move the cursor up.

**open_path**() → None

Open the selected file.

**property path: *Path***

Return the current selected path.

**preferred_height**(*width: int*, *max_available_height: int*, *wrap_lines: bool*, *get_line_prefix: GetLinePrefixCallable* | *None*) → int | None

**preferred_width**(*max_available_width: int*) → int | None

**reset**() → None

**select**(*row: int | None*, *open_file: bool = False*) → NotImplementedOrNone

Select a file in the browser.

euporie.core.widgets.file_browser.**is_dir**(*path: str | pathlib.Path*) → bool | None

Check if a path is a directory.

### euporie.core.widgets.formatted_text_area

Contain dpdated ANSI parsing and Formatted Text processing.

### Functions

| | |
|---|---|
| *fragment_list_to_text*(fragments) | Concatenate all the text parts again. |
| *split_lines*(fragments) | Take a single list of (style_str, text) tuples and yield one such list for each line. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |

### euporie.core.widgets.formatted_text_area.fragment_list_to_text

euporie.core.widgets.formatted_text_area.**fragment_list_to_text**(*fragments: StyleAndTextTuples*) → str

Concatenate all the text parts again.

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.formatted_text_area.split_lines

euporie.core.widgets.formatted_text_area.**split_lines**(*fragments: Iterable[OneStyleAndTextTuple]*) → Iterable[StyleAndTextTuples]

Take a single list of (style_str, text) tuples and yield one such list for each line. Just like str.split, this will yield at least one item.

**Parameters**

**fragments** – Iterable of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.formatted_text_area.to_filter

euporie.core.widgets.formatted_text_area.**to_filter**(*bool_or_filter: Union[*Filter*, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### euporie.core.widgets.formatted_text_area.to_formatted_text

euporie.core.widgets.formatted_text_area.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

## Classes

| | |
|---|---|
| *ConditionalMargin*(margin, filter) | Wrapper around other *Margin* classes to show/hide them. |
| *DynamicProcessor*(get_processor) | Processor class that dynamically returns any Processor. |
| *FormattedTextArea*(formatted_text, *args[, ...]) | Apply formatted text to a TextArea. |
| *FormattedTextProcessor*(formatted_text) | Apply formatted text to a TextArea. |
| *NumberedMargin*([diagnostics, show_diagnostics]) | Margin that displays the line numbers of a `Window`. |
| *Processor*() | Manipulate the fragments for a given line in a *BufferControl*. |
| *TextArea*([text, multiline, password, lexer, ...]) | A simple input field. |
| *Transformation*(fragments[, ...]) | Transformation result, as returned by `Processor.apply_transformation()`. |

### euporie.core.widgets.formatted_text_area.ConditionalMargin

**class** euporie.core.widgets.formatted_text_area.**ConditionalMargin**(*margin: Margin*, *filter: Union[*Filter*, bool]*)

Wrapper around other *Margin* classes to show/hide them.

### euporie.core.widgets.formatted_text_area.DynamicProcessor

**class** euporie.core.widgets.formatted_text_area.**DynamicProcessor**(*get_processor: Callable[[], prompt_toolkit.layout.processors.Processor | None]*)

Processor class that dynamically returns any Processor.

**Parameters**
**get_processor** – Callable that returns a *Processor* instance.

### euporie.core.widgets.formatted_text_area.FormattedTextArea

**class** euporie.core.widgets.formatted_text_area.**FormattedTextArea**(*formatted_text: AnyFormattedText, *args: Any, line_numbers: FilterOrBool = False, **kwargs: Any*)

Apply formatted text to a TextArea.

### euporie.core.widgets.formatted_text_area.FormattedTextProcessor

**class** euporie.core.widgets.formatted_text_area.**FormattedTextProcessor**(*formatted_text: StyleAndTextTuples*)

Apply formatted text to a TextArea.

### euporie.core.widgets.formatted_text_area.NumberedMargin

**class** euporie.core.widgets.formatted_text_area.**NumberedMargin**(*diagnostics: Report | Callable[[], Report] | None = None, show_diagnostics: FilterOrBool = False*)

Margin that displays the line numbers of a `Window`.

### euporie.core.widgets.formatted_text_area.Processor

**class** euporie.core.widgets.formatted_text_area.**Processor**

Manipulate the fragments for a given line in a *BufferControl*.

### euporie.core.widgets.formatted_text_area.TextArea

**class** euporie.core.widgets.formatted_text_area.**TextArea**(*text: str = ''*, *multiline: FilterOrBool = True*, *password: FilterOrBool = False*, *lexer: Lexer | None = None*, *auto_suggest: AutoSuggest | None = None*, *completer: Completer | None = None*, *complete_while_typing: FilterOrBool = True*, *validator: Validator | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *history: History | None = None*, *focusable: FilterOrBool = True*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = True*, *read_only: FilterOrBool = False*, *width: AnyDimension = None*, *height: AnyDimension = None*, *dont_extend_height: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *line_numbers: bool = False*, *get_line_prefix: GetLinePrefixCallable | None = None*, *scrollbar: bool = False*, *style: str = ''*, *search_field: SearchToolbar | None = None*, *preview_search: FilterOrBool = True*, *prompt: AnyFormattedText = ''*, *input_processors: list[Processor] | None = None*, *name: str = ''*)

A simple input field.

This is a higher level abstraction on top of several other classes with sane defaults.

This widget does have the most common options, but it does not intend to cover every single use case. For more configurations options, you can always build a text area manually, using a *Buffer*, BufferControl and Window.

Buffer attributes:

**Parameters**

- **text** – The initial text.
- **multiline** – If True, allow multiline input.

- **completer** – Completer instance for auto completion.
- **complete_while_typing** – Boolean.
- **accept_handler** – Called when *Enter* is pressed (This should be a callable that takes a buffer as input).
- **history** – *History* instance.
- **auto_suggest** – *AutoSuggest* instance for input suggestions.

BufferControl attributes:

**Parameters**

- **password** – When *True*, display using asterisks.
- **focusable** – When *True*, allow this widget to receive the focus.
- **focus_on_click** – When *True*, focus after mouse click.
- **input_processors** – *None* or a list of Processor objects.
- **validator** – *None* or a *Validator* object.

Window attributes:

**Parameters**

- **lexer** – Lexer instance for syntax highlighting.
- **wrap_lines** – When *True*, don't scroll horizontally, but wrap lines.
- **width** – Window width. (Dimension object.)
- **height** – Window height. (Dimension object.)
- **scrollbar** – When *True*, display a scroll bar.
- **style** – A style string.
- **dont_extend_width** – When *True*, don't take up more width then the preferred width reported by the control.
- **dont_extend_height** – When *True*, don't take up more width then the preferred height reported by the control.
- **get_line_prefix** – None or a callable that returns formatted text to be inserted before a line. It takes a line number (int) and a wrap_count and returns formatted text. This can be used for implementation of line continuations, things like Vim "breakindent" and so on.

Other attributes:

**Parameters**

**search_field** – An optional *SearchToolbar* object.

### euporie.core.widgets.formatted_text_area.Transformation

**class** euporie.core.widgets.formatted_text_area.**Transformation**(*fragments: StyleAndTextTuples*, *source_to_display: SourceToDisplay | None = None*, *display_to_source: DisplayToSource | None = None*)

Transformation result, as returned by `Processor.apply_transformation()`.

**Important: Always make sure that the length of *document.text* is equal to** the length of all the text in *fragments*!

**Parameters**

- **fragments** – The transformed fragments. To be displayed, or to pass to the next processor.
- **source_to_display** – Cursor position transformation from original string to transformed string.
- **display_to_source** – Cursor position transformed from source string to original string.

**class** euporie.core.widgets.formatted_text_area.**FormattedTextArea**(*formatted_text: AnyFormattedText*, **args: Any*, *line_numbers: FilterOrBool = False*, ***kwargs: Any*)

Bases: *TextArea*

Apply formatted text to a TextArea.

**property accept_handler: Optional[Callable[[*Buffer*], bool]]**

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

**property document: *Document***

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

**property formatted_text: StyleAndTextTuples**

The formatted text.

**get_processor**() → *FormattedTextProcessor*

Generate a processor for the formatted text.

**property text: str**

The *Buffer* text.

**class** euporie.core.widgets.formatted_text_area.**FormattedTextProcessor**(*formatted_text: StyleAndTextTuples*)

Bases: *Processor*

Apply formatted text to a TextArea.

**apply_transformation**(*transformation_input: TransformationInput*) → *Transformation*

Apply text formatting to a line in a buffer.

### euporie.core.widgets.forms

Contain input widgets.

### Functions

| | |
|---|---|
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |
| *align*(ft[, how, width, style, placeholder, ...]) | Align formatted text at a given width. |
| *cast*(typ, val) | Cast a value to a type. |
| *ceil*(x, /) | Return the ceiling of x as an Integral. |
| *explode_text_fragments*(fragments) | Turn a list of (style_str, text) tuples into another list where each string is exactly one character. |
| *floor*(x, /) | Return the floor of x as an Integral. |
| *fragment_list_len*(fragments) | Return the amount of characters in this text fragment list. |
| *fragment_list_width*(fragments) | Return the character width of this text fragment list. |
| *get_app*() | Get the current active (running) Application. |
| *has_focus*(value) | Enable when this buffer has the focus. |
| *merge_key_bindings*(bindings) | Merge multiple `Keybinding` objects together. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |

### euporie.core.widgets.forms.abstractmethod

euporie.core.widgets.forms.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):

…

### euporie.core.widgets.forms.align

euporie.core.widgets.forms.**align**(*ft: StyleAndTextTuples*, *how:* FormattedTextAlign = *FormattedTextAlign.LEFT*, *width: int | None = None*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Align formatted text at a given width.

**Parameters**

- **how** – The alignment direction
- **ft** – The formatted text to strip
- **width** – The width to which the output should be padded. If `None`, the length of the longest line is used
- **style** – The style to apply to the padding
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – If True, whitespace will be ignored

**Returns**

The aligned formatted text

### euporie.core.widgets.forms.cast

euporie.core.widgets.forms.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.widgets.forms.ceil

euporie.core.widgets.forms.**ceil**(*x*, */*)

Return the ceiling of x as an Integral.

This is the smallest integer >= x.

### euporie.core.widgets.forms.explode_text_fragments

euporie.core.widgets.forms.**explode_text_fragments**(*fragments: Iterable[_T]*) → _ExplodedList[_T]

Turn a list of (style_str, text) tuples into another list where each string is exactly one character.

It should be fine to call this function several times. Calling this on a list that is already exploded, is a null operation.

**Parameters**

**fragments** – List of (style, text) tuples.

### euporie.core.widgets.forms.floor

euporie.core.widgets.forms.**floor**(*x*, */*)

Return the floor of x as an Integral.

This is the largest integer <= x.

### euporie.core.widgets.forms.fragment_list_len

euporie.core.widgets.forms.**fragment_list_len**(*fragments: StyleAndTextTuples*) → int

Return the amount of characters in this text fragment list.

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.forms.fragment_list_width

euporie.core.widgets.forms.**fragment_list_width**(*fragments: StyleAndTextTuples*) → int

Return the character width of this text fragment list. (Take double width characters into account.)

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.forms.get_app

euporie.core.widgets.forms.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.forms.has_focus

euporie.core.widgets.forms.**has_focus**(*value: FocusableElement*) → *Condition*

Enable when this buffer has the focus.

### euporie.core.widgets.forms.merge_key_bindings

euporie.core.widgets.forms.**merge_key_bindings**(*bindings: Sequence[KeyBindingsBase]*) → _MergedKeyBindings

Merge multiple `Keybinding` objects together.

Usage:

```
bindings = merge_key_bindings([bindings1, bindings2, ...])
```

### euporie.core.widgets.forms.to_filter

euporie.core.widgets.forms.**to_filter**(*bool_or_filter: Union[*Filter*, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### euporie.core.widgets.forms.to_formatted_text

euporie.core.widgets.forms.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

### Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *AfterInput*(text[, style]) | Insert text after the input. |
| *Always*() | Always enable feature. |
| *Border*(body, border, style, str] =, show_borders) | Draw a border around any container. |
| *Box*(body[, padding, padding_left, ...]) | Add padding around a container. |
| *BufferControl*([buffer, input_processors, ...]) | Control for visualizing the content of a *Buffer*. |
| *Button*(text, on_click, None] \| None = None, ...) | A clickable button widget. |
| *Checkbox*([text, on_click, prefix, style, ...]) | A toggleable checkbox widget. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalCompleter*(completer, filter) | Wrapper around any other completer that will enable/disable the completions depending on whether the received condition is satisfied. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *ConditionalKeyBindings*(key_bindings[, filter]) | Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.::. |
| *ConditionalProcessor*(processor, filter) | Processor that applies another processor, according to a certain condition. Example::. |
| *ConditionalSplit*(vertical, *args, **kwargs) | A split container where the orientation depends on a filter. |
| *DiBool*([top, right, bottom, left]) | A tuple of four bools with directions. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *Dropdown*(options[, labels, index, indices, ...]) | A dropdown widget, allowing selection of an item from a menu of options. |
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *ExpandingBufferControl*([buffer, ...]) | A sub-class of *BufferControl* which expands to the available width. |

continues on next page

Table 9 – continued from previous page

| | |
|---|---|
| *Filter*() | Base class for any filter to activate/deactivate a feature, depending on a condition. |
| *Float*(content[, top, right, bottom, left, ...]) | Float for use in a *FloatContainer*. |
| *FormattedTextAlign*(value[, names, module, ...]) | Alignment of formatted text. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *KeyBindings*() | A container for a set of key bindings. |
| *Label*(value[, style, html]) | A label widget which displays rich text. |
| *LabelledWidget*(body, label[, style, ...]) | A widget which applies a label to another widget. |
| *MarginContainer*(margin, target) | A container which renders a stand-alone margin. |
| *MouseButton*(value[, names, module, ...]) | |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, sent to *UIControl.mouse_handler*. |
| *MouseEventType*(value[, names, module, ...]) | |
| *NavigableFormattedTextControl*([text, style, ...]) | Formatted text control where the cursor can be moved by scrolling. |
| *Point*(x, y) | |
| *Progress*([start, stop, step, value, ...]) | A progress-bar widget. |
| *ProgressControl*([start, stop, step, value, ...]) | A control which draws a progress-bar. |
| *ScrollbarMargin*([display_arrows, ...]) | Margin displaying a scrollbar. |
| *Select*(options, labels, index, indices, ...) | A select widget, which allows one or more items to be selected from a list. |
| *SelectableWidget*(options[, labels, index, ...]) | Base class for widgets where one or more items can be selected. |
| *Shadow*(body) | Draw a shadow underneath/behind this container. |
| *SimpleCache*([maxsize]) | Very simple cache that discards the oldest item when the cache size is exceeded. |
| *SizedMask*([size]) | Mask with restricted number of True items. |
| *Slider*(options, labels, index, indices, ...) | A slider widget with an optional editable readout. |
| *SliderControl*(slider[, show_arrows, ...]) | A control to display a slider. |
| *Swatch*(color, str] =, width, height, style, ...) | An widget which displays a given color. |
| *Text*([text, style, height, min_height, ...]) | A text input widget. |
| *TextArea*([text, multiline, password, lexer, ...]) | A simple input field. |
| *ToggleButton*(text, on_click, ...) | A toggleable button widget. |
| *ToggleButtons*(options, labels, index, ...) | A widget where an option is selected using mutually exclusive toggle-buttons. |
| *ToggleableWidget*() | Base class for toggleable widgets. |
| *UIContent*(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| *UIControl*() | Base class for all user interface controls. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *ValidationState*(value[, names, module, ...]) | The validation state of a buffer. |
| *Validator*() | Abstract base class for an input validator. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *WordCompleter*(words[, ignore_case, ...]) | Simple autocompletion on a list of words. |
| *WritePosition*(xpos, ypos, width, height) | |
| *deque* | deque([iterable[, maxlen]]) --> deque object |
| *finalize*(obj, func, /, *args, **kwargs) | Class for finalization of weakrefable objects |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.widgets.forms.ABCMeta

**class** euporie.core.widgets.forms.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.widgets.forms.AfterInput

**class** euporie.core.widgets.forms.**AfterInput**(*text: AnyFormattedText*, *style: str = ''*)

Insert text after the input.

**Parameters**

- **text** – This can be either plain text or formatted text (or a callable that returns any of those).
- **style** – style to be applied to this prompt/prefix.

### euporie.core.widgets.forms.Always

**class** euporie.core.widgets.forms.**Always**

Always enable feature.

### euporie.core.widgets.forms.Border

**class** euporie.core.widgets.forms.**Border**(*body: AnyContainer*, *border: GridStyle | None = □□□□ □□ □□ □□□□ □□□□*, *style: str | Callable[[], str] = 'class:border'*, *show_borders: DiBool | None = None*)

Draw a border around any container.

### euporie.core.widgets.forms.Box

**class** euporie.core.widgets.forms.**Box**(*body: AnyContainer*, *padding: AnyDimension = None*, *padding_left: AnyDimension = None*, *padding_right: AnyDimension = None*, *padding_top: AnyDimension = None*, *padding_bottom: AnyDimension = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *char: None | str | Callable[[], str] = None*, *modal: bool = False*, *key_bindings:* KeyBindings *| None = None*)

Add padding around a container.

This also makes sure that the parent can provide more space than required by the child. This is very useful when wrapping a small element with a fixed size into a `VSplit` or `HSplit` object. The `HSplit` and `VSplit` try to make sure to adapt respectively the width and height, possibly shrinking other elements. Wrapping something in a `Box` makes it flexible.

**Parameters**

- **body** – Another container object.
- **padding** – The margin to be used around the body. This can be
- **padding_left** (*overridden by*) – *padding_bottom*.
- **padding_right`** – *padding_bottom*.
- **and** (*padding_top*) – *padding_bottom*.
- **style** – A style string.
- **char** – Character to be used for filling the space around the body. (This is supposed to be a character with a terminal width of 1.)

### euporie.core.widgets.forms.BufferControl

**class** euporie.core.widgets.forms.**BufferControl**(*buffer:* Buffer | *None* = *None*, *input_processors:* *list[*Processor*]* | *None* = *None*, *include_default_input_processors:* *bool* = *True*, *lexer:* *Lexer* | *None* = *None*, *preview_search:* *FilterOrBool* = *False*, *focusable:* *FilterOrBool* = *True*, *search_buffer_control:* *None* | SearchBufferControl | *Callable[[],* SearchBufferControl*]* = *None*, *menu_position:* *Callable[[],* *int* | *None]* | *None* = *None*, *focus_on_click:* *FilterOrBool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*)

Control for visualizing the content of a *Buffer*.

**Parameters**

- **buffer** – The *Buffer* object to be displayed.
- **input_processors** – A list of *Processor* objects.
- **include_default_input_processors** – When True, include the default processors for highlighting of selection, search and displaying of multiple cursors.
- **lexer** – Lexer instance for syntax highlighting.
- **preview_search** – *bool* or *Filter*: Show search while typing. When this is *True*, probably you want to add a HighlightIncrementalSearchProcessor as well. Otherwise only the cursor position will move, but the text won't be highlighted.
- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **focus_on_click** – Focus this buffer when it's click, but not yet focused.
- **key_bindings** – a *KeyBindings* object.

### euporie.core.widgets.forms.Button

**class** euporie.core.widgets.forms.**Button**(*text: AnyFormattedText, on_click: Callable[[Button], None] | None = None, on_mouse_down: Callable[[Button], None] | None = None, disabled: FilterOrBool = False, width: int | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None, selected: bool = False, key_bindings: KeyBindingsBase | None = None, mouse_handler: Callable[[MouseEvent], NotImplementedOrNone] | None = None*)

A clickable button widget.

### euporie.core.widgets.forms.Checkbox

**class** euporie.core.widgets.forms.**Checkbox**(*text: AnyFormattedText = '', on_click: Callable[[ToggleableWidget], None] | None = None, prefix: tuple[str, str] = ('□', '□'), style: str = 'class:input', selected: bool = False, disabled: FilterOrBool = False, key_bindings: KeyBindingsBase | None = None*)

A toggleable checkbox widget.

### euporie.core.widgets.forms.Condition

**class** euporie.core.widgets.forms.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.forms.ConditionalCompleter

**class** euporie.core.widgets.forms.**ConditionalCompleter**(*completer: Completer, filter: Union[Filter, bool]*)

Wrapper around any other completer that will enable/disable the completions depending on whether the received condition is satisfied.

**Parameters**

- **completer** – *Completer* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.forms.ConditionalContainer

**class** euporie.core.widgets.forms.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.forms.ConditionalKeyBindings

**class** euporie.core.widgets.forms.**ConditionalKeyBindings**(*key_bindings: KeyBindingsBase*, *filter: Union[Filter, bool] = True*)

Wraps around a *KeyBindings*. Disable/enable all the key bindings according to the given (additional) filter.:

```
@Condition
def setting_is_true():
    return True  # or False

registry = ConditionalKeyBindings(key_bindings, setting_is_true)
```

When new key bindings are added to this object. They are also enable/disabled according to the given *filter*.

**Parameters**

- **registries** – List of *KeyBindings* objects.
- **filter** – Filter object.

### euporie.core.widgets.forms.ConditionalProcessor

**class** euporie.core.widgets.forms.**ConditionalProcessor**(*processor: Processor*, *filter: Union[Filter, bool]*)

Processor that applies another processor, according to a certain condition. Example:

```
# Create a function that returns whether or not the processor should
# currently be applied.
def highlight_enabled():
    return true_or_false

# Wrapped it in a `ConditionalProcessor` for usage in a `BufferControl`.
BufferControl(input_processors=[
    ConditionalProcessor(HighlightSearchProcessor(),
                         Condition(highlight_enabled))])
```

**Parameters**

- **processor** – *Processor* instance.
- **filter** – Filter instance.

### euporie.core.widgets.forms.ConditionalSplit

**class** euporie.core.widgets.forms.**ConditionalSplit**(*vertical: FilterOrBool*, **args: Any*, ***kwargs: Any*)

A split container where the orientation depends on a filter.

### euporie.core.widgets.forms.DiBool

**class** euporie.core.widgets.forms.**DiBool**(*top: bool = False*, *right: bool = False*, *bottom: bool = False*, *left: bool = False*)

A tuple of four bools with directions.

### euporie.core.widgets.forms.Dimension

**class** euporie.core.widgets.forms.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.widgets.forms.Dropdown

**class** euporie.core.widgets.forms.**Dropdown**(*options: list[Any]*, *labels: Sequence[AnyFormattedText] | None = None*, *index: int | None = None*, *indices: list[int] | None = None*, *n_values: int | None = None*, *multiple: FilterOrBool = False*, *max_count: int | None = None*, *on_change: Callable[[SelectableWidget], None] | None = None*, *style: str | Callable[[], str] = 'class:input'*, *arrow: str = '▽'*, *disabled: FilterOrBool = False*)

A dropdown widget, allowing selection of an item from a menu of options.

### euporie.core.widgets.forms.Event

**class** euporie.core.widgets.forms.**Event**(*sender: _Sender*, *handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.widgets.forms.ExpandingBufferControl

**class** euporie.core.widgets.forms.**ExpandingBufferControl**(*buffer: Buffer | None = None*, *input_processors: list[Processor] | None = None*, *include_default_input_processors: bool = True*, *lexer: Lexer | None = None*, *preview_search: FilterOrBool = False*, *focusable: FilterOrBool = True*, *search_buffer_control: OptionalSearchBuffer = None*, *menu_position: Callable[[], int | None] | None = None*, *focus_on_click: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *expand: FilterOrBool = True*)

A sub-class of *BufferControl* which expands to the available width.

### euporie.core.widgets.forms.Filter

**class** euporie.core.widgets.forms.**Filter**

Base class for any filter to activate/deactivate a feature, depending on a condition.

The return value of __call__ will tell if the feature should be active.

### euporie.core.widgets.forms.Float

**class** euporie.core.widgets.forms.**Float**(*content: AnyContainer*, *top: int | None = None*, *right: int | None = None*, *bottom: int | None = None*, *left: int | None = None*, *width: int | Callable[[], int] | None = None*, *height: int | Callable[[], int] | None = None*, *xcursor: bool = False*, *ycursor: bool = False*, *attach_to_window: AnyContainer | None = None*, *hide_when_covering_content: bool = False*, *allow_cover_cursor: bool = False*, *z_index: int = 1*, *transparent: bool = False*)

Float for use in a *FloatContainer*. Except for the *content* parameter, all other options are optional.

**Parameters**

- **content** – *Container* instance.
- **width** – *Dimension* or callable which returns a *Dimension*.
- **height** – *Dimension* or callable which returns a *Dimension*.
- **left** – Distance to the left edge of the *FloatContainer*.
- **right** – Distance to the right edge of the *FloatContainer*.
- **top** – Distance to the top of the *FloatContainer*.
- **bottom** – Distance to the bottom of the *FloatContainer*.
- **attach_to_window** – Attach to the cursor from this window, instead of the current window.
- **hide_when_covering_content** – Hide the float when it covers content underneath.
- **allow_cover_cursor** – When *False*, make sure to display the float below the cursor. Not on top of the indicated position.
- **z_index** – Z-index position. For a Float, this needs to be at least one. It is relative to the z_index of the parent container.
- **transparent** – *Filter* indicating whether this float needs to be drawn transparently.

### euporie.core.widgets.forms.FormattedTextAlign

**class** euporie.core.widgets.forms.**FormattedTextAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of formatted text.

### euporie.core.widgets.forms.FormattedTextControl

**class** euporie.core.widgets.forms.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[],* Point | *None]* | *None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **key_bindings** – a *KeyBindings* object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.widgets.forms.KeyBindings

**class** euporie.core.widgets.forms.**KeyBindings**

A container for a set of key bindings.

Example usage:

```
kb = KeyBindings()

@kb.add('c-t')
def _(event):
    print('Control-T pressed')

@kb.add('c-a', 'c-b')
def _(event):
```

(continues on next page)

(continued from previous page)

```
    print('Control-A pressed, followed by Control-B')

@kb.add('c-x', filter=is_searching)
def _(event):
    print('Control-X pressed')  # Works only if we are searching.
```

### euporie.core.widgets.forms.Label

**class** euporie.core.widgets.forms.**Label**(*value: AnyFormattedText*, *style: str | Callable[[], str] = 'class:input'*, *html: FilterOrBool = False*)

A label widget which displays rich text.

### euporie.core.widgets.forms.LabelledWidget

**class** euporie.core.widgets.forms.**LabelledWidget**(*body: AnyContainer*, *label: AnyFormattedText*, *style: str = 'class:input'*, *vertical: FilterOrBool = False*, *html: FilterOrBool = False*)

A widget which applies a label to another widget.

### euporie.core.widgets.forms.MarginContainer

**class** euporie.core.widgets.forms.**MarginContainer**(*margin:* Margin, *target: ScrollableContainer*)

A container which renders a stand-alone margin.

### euporie.core.widgets.forms.MouseButton

**class** euporie.core.widgets.forms.**MouseButton**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.forms.MouseEvent

**class** euporie.core.widgets.forms.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers: frozenset[*prompt_toolkit.mouse_events.MouseModifier*]*)

Mouse event, sent to *UIControl.mouse_handler*.

**Parameters**

- **position** – *Point* instance.
- **event_type** – *MouseEventType*.

### euporie.core.widgets.forms.MouseEventType

**class** euporie.core.widgets.forms.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.forms.NavigableFormattedTextControl

**class** euporie.core.widgets.forms.**NavigableFormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*, *move_cursor_up: Callable[[], None] | None = None*, *move_cursor_down: Callable[[], None] | None = None*)

Formatted text control where the cursor can be moved by scrolling.

### euporie.core.widgets.forms.Point

**class** euporie.core.widgets.forms.**Point**(*x*, *y*)

### euporie.core.widgets.forms.Progress

**class** euporie.core.widgets.forms.**Progress**(*start: float | int = 0*, *stop: float | int = 100*, *step: float | int = 1*, *value: float | int = 0*, *vertical: FilterOrBool = False*, *style: str | Callable[[], str] = 'class:input'*)

A progress-bar widget.

### euporie.core.widgets.forms.ProgressControl

**class** euporie.core.widgets.forms.**ProgressControl**(*start: float | int = 0*, *stop: float | int = 100*, *step: float | int = 1*, *value: float | int = 0*, *vertical: Union[Filter, bool] = False*)

A control which draws a progress-bar.

### euporie.core.widgets.forms.ScrollbarMargin

**class** euporie.core.widgets.forms.**ScrollbarMargin**(*display_arrows: Union[Filter, bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[Filter, bool] = False*, *smooth: bool = True*, *style: str = ''*)

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

### euporie.core.widgets.forms.Select

**class** euporie.core.widgets.forms.**Select**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input,select', rows: int | None = 3, prefix: tuple[str, str] = ('', ''), border: GridStyle | None = □□ □ □□ □□□□□ □□ , show_borders: DiBool | None = None, disabled: FilterOrBool = False, dont_extend_width: FilterOrBool = True, dont_extend_height: FilterOrBool = True*)

A select widget, which allows one or more items to be selected from a list.

### euporie.core.widgets.forms.SelectableWidget

**class** euporie.core.widgets.forms.**SelectableWidget**(*options: list[Any]*, *labels: Sequence[AnyFormattedText] | None = None*, *index: int | None = None*, *indices: list[int] | None = None*, *n_values: int | None = None*, *multiple: FilterOrBool = False*, *max_count: int | None = None*, *on_change: Callable[[SelectableWidget], None] | None = None*, *style: str | Callable[[], str] = 'class:input'*, *disabled: FilterOrBool = False*)

Base class for widgets where one or more items can be selected.

### euporie.core.widgets.forms.Shadow

**class** euporie.core.widgets.forms.**Shadow**(*body: AnyContainer*)

Draw a shadow underneath/behind this container.

This is a globally configurable version of the `prompt_toolkit.widows.base.Shadow` class.

### euporie.core.widgets.forms.SimpleCache

**class** euporie.core.widgets.forms.**SimpleCache**(*maxsize: int = 8*)

Very simple cache that discards the oldest item when the cache size is exceeded.

**Parameters**

**maxsize** – Maximum size of the cache. (Don't make it too big.)

### euporie.core.widgets.forms.SizedMask

**class** euporie.core.widgets.forms.**SizedMask**(*size: int | None = None*)

Mask with restricted number of True items.

### euporie.core.widgets.forms.Slider

**class** euporie.core.widgets.forms.**Slider**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle = ▯▯ ▯ ▯▯ ▯▯▯▯ ▯▯, show_borders: DiBool | None = None, vertical: FilterOrBool = False, show_arrows: FilterOrBool = True, arrows: tuple[AnyFormattedText, AnyFormattedText] = ('-', '+'), show_readout: FilterOrBool = True, disabled: FilterOrBool = False*)

A slider widget with an optional editable readout.

### euporie.core.widgets.forms.SliderControl

**class** euporie.core.widgets.forms.**SliderControl**(*slider: Slider, show_arrows: Union[Filter, bool] = True, handle_char: str = '▯', track_char: str | None = None, selected_track_char: str | None = None, style: str = 'class:input', disabled: Union[Filter, bool] = False*)

A control to display a slider.

### euporie.core.widgets.forms.Swatch

**class** euporie.core.widgets.forms.**Swatch**(*color: str | Callable[[], str] = '#FFFFFF', width: int = 2, height: int = 1, style: str = 'class:swatch', border: GridStyle = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None*)

An widget which displays a given color.

### euporie.core.widgets.forms.Text

**class** euporie.core.widgets.forms.**Text**(*text: str = '', style: str = 'class:input', height: int = 1, min_height: int = 1, multiline: FilterOrBool = False, expand: FilterOrBool = True, width: int | None = None, completer: Completer | None = None, options: list[str] | Callable[[], list[str]] | None = None, show_borders: DiBool | None = None, on_text_changed: Callable[[Buffer], None] | None = None, validation: Callable[[str], bool] | None = None, accept_handler: BufferAcceptHandler | None = None, placeholder: str | None = None, lexer: Lexer | None = None, input_processors: Sequence[Processor] | None = None, disabled: FilterOrBool = False, password: FilterOrBool = False, wrap_lines: FilterOrBool = False, prompt: AnyFormattedText | None = None*)

A text input widget.

### euporie.core.widgets.forms.TextArea

**class** euporie.core.widgets.forms.**TextArea**(*text: str = '', multiline: FilterOrBool = True, password: FilterOrBool = False, lexer: Lexer | None = None, auto_suggest: AutoSuggest | None = None, completer: Completer | None = None, complete_while_typing: FilterOrBool = True, validator: Validator | None = None, accept_handler: BufferAcceptHandler | None = None, history: History | None = None, focusable: FilterOrBool = True, focus_on_click: FilterOrBool = False, wrap_lines: FilterOrBool = True, read_only: FilterOrBool = False, width: AnyDimension = None, height: AnyDimension = None, dont_extend_height: FilterOrBool = False, dont_extend_width: FilterOrBool = False, line_numbers: bool = False, get_line_prefix: GetLinePrefixCallable | None = None, scrollbar: bool = False, style: str = '', search_field: SearchToolbar | None = None, preview_search: FilterOrBool = True, prompt: AnyFormattedText = '', input_processors: list[Processor] | None = None, name: str = ''*)

A simple input field.

This is a higher level abstraction on top of several other classes with sane defaults.

This widget does have the most common options, but it does not intend to cover every single use case. For more configurations options, you can always build a text area manually, using a *Buffer*, BufferControl and Window.

Buffer attributes:

**Parameters**

- **text** – The initial text.
- **multiline** – If True, allow multiline input.
- **completer** – `Completer` instance for auto completion.
- **complete_while_typing** – Boolean.
- **accept_handler** – Called when *Enter* is pressed (This should be a callable that takes a buffer as input).
- **history** – *`History`* instance.
- **auto_suggest** – *`AutoSuggest`* instance for input suggestions.

BufferControl attributes:

**Parameters**

- **password** – When *True*, display using asterisks.
- **focusable** – When *True*, allow this widget to receive the focus.
- **focus_on_click** – When *True*, focus after mouse click.
- **input_processors** – *None* or a list of `Processor` objects.
- **validator** – *None* or a *`Validator`* object.

Window attributes:

**Parameters**

- **lexer** – `Lexer` instance for syntax highlighting.
- **wrap_lines** – When *True*, don't scroll horizontally, but wrap lines.
- **width** – Window width. (`Dimension` object.)
- **height** – Window height. (`Dimension` object.)
- **scrollbar** – When *True*, display a scroll bar.
- **style** – A style string.
- **dont_extend_width** – When *True*, don't take up more width then the preferred width reported by the control.
- **dont_extend_height** – When *True*, don't take up more width then the preferred height reported by the control.
- **get_line_prefix** – None or a callable that returns formatted text to be inserted before a line. It takes a line number (int) and a wrap_count and returns formatted text. This can be used for implementation of line continuations, things like Vim "breakindent" and so on.

Other attributes:

**Parameters**

**search_field** – An optional *SearchToolbar* object.

### euporie.core.widgets.forms.ToggleButton

**class** euporie.core.widgets.forms.**ToggleButton**(*text: AnyFormattedText, on_click: Callable[[ToggleButton], None] | None = None, width: int | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None, selected: bool = False, disabled: FilterOrBool = False, key_bindings: KeyBindingsBase | None = None*)

A toggleable button widget.

### euporie.core.widgets.forms.ToggleButtons

**class** euporie.core.widgets.forms.**ToggleButtons**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, disabled: FilterOrBool = False, vertical: FilterOrBool = False*)

A widget where an option is selected using mutually exclusive toggle-buttons.

### euporie.core.widgets.forms.ToggleableWidget

**class** euporie.core.widgets.forms.**ToggleableWidget**

Base class for toggleable widgets.

### euporie.core.widgets.forms.UIContent

**class** euporie.core.widgets.forms.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>, line_count: int = 0, cursor_position: Point | None = None, menu_position: Point | None = None, show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.widgets.forms.UIControl

**class** euporie.core.widgets.forms.**UIControl**

Base class for all user interface controls.

### euporie.core.widgets.forms.VSplit

**class** euporie.core.widgets.forms.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: HorizontalAlign* = *HorizontalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.forms.ValidationState

**class** euporie.core.widgets.forms.**ValidationState**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

The validation state of a buffer. This is set after the validation.

### euporie.core.widgets.forms.Validator

**class** euporie.core.widgets.forms.**Validator**

Abstract base class for an input validator.

A validator is typically created in one of the following two ways:

- Either by overriding this class and implementing the *validate* method.
- Or by passing a callable to *Validator.from_callable*.

If the validation takes some time and needs to happen in a background thread, this can be wrapped in a `ThreadedValidator`.

### euporie.core.widgets.forms.Window

**class** euporie.core.widgets.forms.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.core.widgets.forms.WordCompleter

**class** euporie.core.widgets.forms.**WordCompleter**(*words: list[str] | Callable[[], list[str]]*, *ignore_case: bool = False*, *display_dict: Mapping[str, AnyFormattedText] | None = None*, *meta_dict: Mapping[str, AnyFormattedText] | None = None*, *WORD: bool = False*, *sentence: bool = False*, *match_middle: bool = False*, *pattern: Pattern[str] | None = None*)

Simple autocompletion on a list of words.

**Parameters**

- **words** – List of words or callable that returns a list of words.
- **ignore_case** – If True, case-insensitive completion.
- **meta_dict** – Optional dict mapping words to their meta-text. (This should map strings to strings or formatted text.)
- **WORD** – When True, use WORD characters.
- **sentence** – When True, don't complete by comparing the word before the cursor, but by comparing all the text before the cursor. In this case, the list of words is just a list of strings, where each string can contain spaces. (Can not be used together with the WORD option.)
- **match_middle** – When True, match not only the start, but also in the middle of the word.
- **pattern** – Optional compiled regex for finding the word before the cursor to complete. When given, use this regex pattern instead of default one (see document._FIND_WORD_RE)

### euporie.core.widgets.forms.WritePosition

**class** euporie.core.widgets.forms.**WritePosition**(*xpos: int*, *ypos: int*, *width: int*, *height: int*)

### euporie.core.widgets.forms.deque

**class** euporie.core.widgets.forms.**deque**

deque([iterable[, maxlen]]) –> deque object

A list-like sequence optimized for data accesses near its endpoints.

### euporie.core.widgets.forms.finalize

**class** euporie.core.widgets.forms.**finalize**(*obj*, *func*, */*, **args*, ***kwargs*)

Class for finalization of weakrefable objects

finalize(obj, func, *args, **kwargs) returns a callable finalizer object which will be called when obj is garbage collected. The first time the finalizer is called it evaluates func(*arg, **kwargs) and returns the result. After this the finalizer is dead, and calling it just returns None.

When the program exits any remaining finalizers for which the atexit attribute is true will be run in reverse order of creation. By default atexit is true.

### euporie.core.widgets.forms.partial

**class** euporie.core.widgets.forms.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.widgets.forms.**Button**(*text: AnyFormattedText, on_click: Callable[[Button], None] | None = None, on_mouse_down: Callable[[Button], None] | None = None, disabled: FilterOrBool = False, width: int | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None, selected: bool = False, key_bindings: KeyBindingsBase | None = None, mouse_handler: Callable[[MouseEvent], NotImplementedOrNone] | None = None*)

Bases: object

A clickable button widget.

**default_mouse_handler**(*mouse_event: MouseEvent*) → NotImplementedOrNone

Handle mouse events.

**get_key_bindings**() → *KeyBindingsBase*

Key bindings for the Button.

**get_style**() → str

Return the style for the button given its current state.

**get_text_fragments**() → StyleAndTextTuples

Return the list of formatted text fragments which define the button.

**property width: int**

The width of the button.

**class** euporie.core.widgets.forms.**Checkbox**(*text: AnyFormattedText = ''*, *on_click: Callable[[ToggleableWidget], None] | None = None*, *prefix: tuple[str, str] = ('', '')*, *style: str = 'class:input'*, *selected: bool = False*, *disabled: FilterOrBool = False*, *key_bindings:* KeyBindingsBase *| None = None*)

Bases: *ToggleableWidget*

A toggleable checkbox widget.

**container: AnyContainer**

**disabled: *Filter***

**key_bindings: *KeyBindingsBase* | None**

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Focus on mouse down and toggle state on mouse up.

**on_click: *Event***

**selected: bool**

**toggle**() → None

Toggle the selected state and trigger the "clicked" callback.

**class** euporie.core.widgets.forms.**Dropdown**(*options: list[Any]*, *labels: Sequence[AnyFormattedText] | None = None*, *index: int | None = None*, *indices: list[int] | None = None*, *n_values: int | None = None*, *multiple: FilterOrBool = False*, *max_count: int | None = None*, *on_change: Callable[[SelectableWidget], None] | None = None*, *style: str | Callable[[], str] = 'class:input'*, *arrow: str = ''*, *disabled: FilterOrBool = False*)

Bases: *SelectableWidget*

A dropdown widget, allowing selection of an item from a menu of options.

**button_text**() → StyleAndTextTuples

Return the text to display on the button.

**hover_rel**(*rel: int*) → None

Hover an index relative to the current hovered index.

**property index: int | None**

Return the first selected index.

**property indices: list[int]**

Return a list of the selected indices.

**key_bindings**() → *KeyBindingsBase*

Return key-bindings for the drop-down widget.

**load_container**() → AnyContainer

Load the widget's container.

**property mask: list[bool]**

Get mask of selected options.

**menu_fragments**() → StyleAndTextTuples

Return formatted text fragment to display in the menu.

**mouse_handler**(*i: int*, *mouse_event:* MouseEvent) → None

Handle mouse events.

**select_rel**(*rel: int*) → None

Select an index relative to the current index.

**property style: str**

Return the widget's style.

**toggle_item**(*index: int*) → None

Toggle the selection status of the option at a given index.

**toggle_menu**(*button:* ToggleButton) → None

Show or hide the menu.

**property value: Any**

Return the selected value.

**property values: list[Any]**

Return a list of the selected values.

**class** euporie.core.widgets.forms.**ExpandingBufferControl**(*buffer:* Buffer | *None* = *None*, *input_processors: list[*Processor*]* | *None* = *None*, *include_default_input_processors: bool* = *True*, *lexer: Lexer* | *None* = *None*, *preview_search: FilterOrBool* = *False*, *focusable: FilterOrBool* = *True*, *search_buffer_control: OptionalSearchBuffer* = *None*, *menu_position: Callable[[], int* | *None]* | *None* = *None*, *focus_on_click: FilterOrBool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *expand: FilterOrBool* = *True*)

Bases: *BufferControl*

A sub-class of *BufferControl* which expands to the available width.

**create_content**(*width: int*, *height: int*, *preview_search: bool* = *False*) → *UIContent*

Create a UIContent.

**get_invalidate_events**() → Iterable[*Event*[object]]

Return the Window invalidate events.

**get_key_bindings**() → *KeyBindingsBase* | None

When additional key bindings are given. Return these.

**is_focusable**() → bool

Tell whether this user control is focusable.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Mouse handler for this control.

**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: GetLinePrefixCallable | None*) → int | None

**preferred_width**(*max_available_width: int*) → int | None

Enure text box expands to available width.

**Parameters**

**max_available_width** – The maximum available width

**Returns**

The desired width, which is the maximum available

**reset**() → None

**property search_buffer: *prompt_toolkit.buffer.Buffer* | None**

**property search_buffer_control: *prompt_toolkit.layout.controls.SearchBufferControl* | None**

**property search_state: SearchState**

Return the *SearchState* for searching this *BufferControl*. This is always associated with the search control. If one search bar is used for searching multiple *BufferControls*, then they share the same *SearchState*.

**class** euporie.core.widgets.forms.**Label**(*value: AnyFormattedText*, *style: str | Callable[[], str] = 'class:input'*, *html: FilterOrBool = False*)

Bases: object

A label widget which displays rich text.

**get_value**() → AnyFormattedText

Return the current value of the label, converting to formatted text.

**class** euporie.core.widgets.forms.**LabelledWidget**(*body: AnyContainer*, *label: AnyFormattedText*, *style: str = 'class:input'*, *vertical: FilterOrBool = False*, *html: FilterOrBool = False*)

Bases: object

A widget which applies a label to another widget.

**property html: *Filter***

Get the HTML filter value.

**class** euporie.core.widgets.forms.**NavigableFormattedTextControl**(*text: AnyFormattedText = '', style: str = '', focusable: FilterOrBool = False, key_bindings: KeyBindingsBase | None = None, show_cursor: bool = True, modal: bool = False, get_cursor_position: Callable[[], Point | None] | None = None, move_cursor_up: Callable[[], None] | None = None, move_cursor_down: Callable[[], None] | None = None*)

Bases: *FormattedTextControl*

Formatted text control where the cursor can be moved by scrolling.

**create_content**(*width: int*, *height: int | None*) → *UIContent*

Generate the content for this user control.

Returns a *UIContent* instance.

**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**() → *KeyBindingsBase* | None

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.

**is_modal**() → bool

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

(When the fragment list contained mouse handlers and the user clicked on on any of these, the matching handler is called. This handler can still return *NotImplemented* in case we want the Window to handle this particular event.)

**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: GetLinePrefixCallable | None*) → int | None

Return the preferred height for this control.

**preferred_width**(*max_available_width: int*) → int

Return the preferred width for this control. That is the width of the longest line.

**reset**() → None

**class** euporie.core.widgets.forms.**Progress**(*start: float | int = 0*, *stop: float | int = 100*, *step: float | int = 1*, *value: float | int = 0*, *vertical: FilterOrBool = False*, *style: str | Callable[[], str] = 'class:input'*)

Bases: object

A progress-bar widget.

**add_style**(*extra: str*) → Callable[[], str]

Add an additional style to the widget's base style.

**property value: float | int**

Return the current value of the proegress-bar.

**class** euporie.core.widgets.forms.**ProgressControl**(*start: float | int = 0*, *stop: float | int = 100*, *step: float | int = 1*, *value: float | int = 0*, *vertical: Union[*Filter, *bool] = False*)

Bases: *UIControl*

A control which draws a progress-bar.

**create_content**(*width: int*, *height: int*) → *UIContent*

Get or render content for a given output size.

**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**() → *KeyBindingsBase* | None

The key bindings that are specific for this user control.

Return a *KeyBindings* object if some key bindings are specified, or *None* otherwise.

**hchars = ('', '▏', '▎', '▍', '▌', '▋', '▊', '▉')**

**is_focusable**() → bool

Tell whether this user control is focusable.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

When *NotImplemented* is returned, it means that the given event is not handled by the *UIControl* itself. The *Window* or key bindings can decide to handle this event as scrolling or changing focus.

**Parameters**

**mouse_event** – *MouseEvent* instance.

**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: GetLinePrefixCallable | None*) → int | None

Determine the height of the progress-bar depending on its orientation.

**preferred_width**(*max_available_width: int*) → int | None

Determine the width of the progress-bar depending on its orientation.

**render**(*width: int*, *height: int*) → list[StyleAndTextTuples]

Render the progressbar at a given size as lines of formatted text.

**reset**() → None

**vchars = ('', '▁', '▂', '▃', '▄', '▅', '▆', '▇', '█')**

**class** euporie.core.widgets.forms.**Select**(*options: list[Any]*, *labels: Sequence[AnyFormattedText] | None = None*, *index: int | None = None*, *indices: list[int] | None = None*, *n_values: int | None = None*, *multiple: FilterOrBool = False*, *max_count: int | None = None*, *on_change: Callable[[SelectableWidget], None] | None = None*, *style: str | Callable[[], str] = 'class:input,select'*, *rows: int | None = 3*, *prefix: tuple[str, str] = ('', ')*, *border: GridStyle | None = □□ □ □□ □□□□□ □□*, *show_borders: DiBool | None = None*, *disabled: FilterOrBool = False*, *dont_extend_width: FilterOrBool = True*, *dont_extend_height: FilterOrBool = True*)

Bases: *SelectableWidget*

A select widget, which allows one or more items to be selected from a list.

**hover_rel**(*rel: int*) → None

Hover an index relative to the current hovered index.

**property index: int | None**

Return the first selected index.

**property indices: list[int]**

Return a list of the selected indices.

**key_bindings**() → *KeyBindingsBase*

Key bindings for the selectable widget.

**load_container**() → AnyContainer

Load the widget's container.

**property mask: list[bool]**

Get mask of selected options.

**mouse_handler**(*i: int*, *mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

**select_rel**(*rel: int*) → None

Select an index relative to the current index.

**property style: str**

Return the widget's style.

**text_fragments**() → StyleAndTextTuples

Create a list of formatted text fragments to display.

**toggle_item**(*index: int*) → None

Toggle the selection status of the option at a given index.

**property value: Any**

Return the selected value.

**property values: list[Any]**

Return a list of the selected values.

**class** euporie.core.widgets.forms.**SelectableWidget**(*options: list[Any]*, *labels: Sequence[AnyFormattedText] | None = None*, *index: int | None = None*, *indices: list[int] | None = None*, *n_values: int | None = None*, *multiple: FilterOrBool = False*, *max_count: int | None = None*, *on_change: Callable[[SelectableWidget], None] | None = None*, *style: str | Callable[[], str] = 'class:input'*, *disabled: FilterOrBool = False*)

Bases: object

Base class for widgets where one or more items can be selected.

**hover_rel**(*rel: int*) → None

Hover an index relative to the current hovered index.

**property index: int | None**

Return the first selected index.

**property indices: list[int]**

Return a list of the selected indices.

**key_bindings**() → *KeyBindingsBase*

Key bindings for the selectable widget.

**abstract load_container**() → AnyContainer

Abstract method for loading the widget's container.

**property mask: list[bool]**

Get mask of selected options.

**mouse_handler**(*i: int*, *mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

**select_rel**(*rel: int*) → None

Select an index relative to the current index.

**property style: str**

Return the widget's style.

**toggle_item**(*index: int*) → None

Toggle the selection status of the option at a given index.

**property value: Any**

Return the selected value.

**property values: list[Any]**

Return a list of the selected values.

**class** euporie.core.widgets.forms.**SizedMask**(*size: int | None = None*)

Bases: Dict[int, bool]

Mask with restricted number of True items.

**clear**() → None

Clear the dict's items.

**copy**() → a shallow copy of D

**fromkeys**(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

**get**(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

**items**() → a set-like object providing a view on D's items

**keys**() → a set-like object providing a view on D's keys

**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If the key is not found, return the default if given; otherwise, raise a KeyError.

**popitem**()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

**setdefault**(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**update**([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values**() → an object providing a view on D's values

**class** euporie.core.widgets.forms.**Slider**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle = ▒▒ ▒ ▒▒ ▒▒▒▒▒ ▒▒, show_borders: DiBool | None = None, vertical: FilterOrBool = False, show_arrows: FilterOrBool = True, arrows: tuple[AnyFormattedText, AnyFormattedText] = ('-', '+'), show_readout: FilterOrBool = True, disabled: FilterOrBool = False*)

Bases: *SelectableWidget*

A slider widget with an optional editable readout.

**accept_handler**(*buffer:* Buffer) → bool

Set the index to the value(s) entered in the readout buffer.

**control**: *SliderControl*

**hover_rel**(*rel: int*) → None

Hover an index relative to the current hovered index.

**property index: int | None**

Return the first selected index.

**property indices: list[int]**

Return a list of the selected indices.

**key_bindings**() → *KeyBindingsBase*

Key bindings for the selectable widget.

**load_container**() → AnyContainer

Build the slider's container.

**property mask: list[bool]**

Get mask of selected options.

**mouse_handler**(*i: int*, *mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

**readout: *Text***

**readout_len**() → int

Return the length of the readout text area.

**readout_text**(*indices: list[int]*) → str

Return the readout text area value.

**select_rel**(*rel: int*) → None

Select an index relative to the current index.

**property style: str**

Return the widget's style.

**toggle_item**(*index: int*) → None

Toggle the selection status of the option at a given index.

**validate_readout**(*text: str*) → list[Any] | None

Confirm the value entered in the readout is value.

**property value: Any**

Return the selected value.

**value_changed**(*slider:* euporie.core.widgets.forms.SelectableWidget | *None* = *None*) → None

Set the readout text when the slider value changes.

**property values: list[Any]**

Return a list of the selected values.

**class** euporie.core.widgets.forms.**SliderControl**(*slider:* Slider, *show_arrows: Union[*Filter, *bool]* = *True*, *handle_char: str* = '', *track_char: str* | *None* = *None*, *selected_track_char: str* | *None* = *None*, *style: str* = *'class:input'*, *disabled: Union[*Filter, *bool]* = *False*)

Bases: *UIControl*

A control to display a slider.

**create_content**(*width: int*, *height: int*) → *UIContent*

Create an cache the rendered control fragments.

**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**() → *KeyBindingsBase* | None

Key bindings for the Slider.

**is_focusable**() → bool

Tell whether this user control is focusable.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events given the slider's orientation.

**mouse_handler_**(*mouse_event:* MouseEvent, *loc: int*) → NotImplementedOrNone

Handle mouse events.

**mouse_handler_arrow**(*mouse_event:* MouseEvent, *n: int = 0*) → NotImplementedOrNone

Handle mouse events on the slider's arrows.

**mouse_handler_handle**(*mouse_event:* MouseEvent, *handle: int = 0*) → NotImplementedOrNone

Handle mouse events on the slider's handles.

**mouse_handler_scroll**(*mouse_event:* MouseEvent, *handle: int | None = None*) → NotImplementedOrNone

Handle mouse scroll events.

**mouse_handler_track**(*mouse_event:* MouseEvent, *index: int = 0*) → NotImplementedOrNone

Handle mouse events on the slider track.

**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: GetLinePrefixCallable | None*) → int | None

Return the preferred height of the slider control given its orientation.

**preferred_width**(*max_available_width: int*) → int | None

Return the preferred width of the slider control given its orientation.

**render_lines**(*width: int*, *height: int*) → list[StyleAndTextTuples]

Generate formatted text fragments to display the slider.

**async repeat**(*mouse_event:* MouseEvent, *handler: Callable[..., NotImplementedOrNone]*, *timeout: float = 0.25*, ***kwargs: Any*) → None

Repeat a mouse event after a timeout.

**reset**() → None

**property selected_handle: int**

Return the currently selected slider handle.

**set_index**(*handle: int = 0*, *ab: int | None = None*, *rel: int | None = None*, *fire: bool = True*) → NotImplementedOrNone

Set the selected index of the slider.

**class** euporie.core.widgets.forms.**Swatch**(*color: str | Callable[[], str] = '#FFFFFF'*, *width: int = 2*, *height: int = 1*, *style: str = 'class:swatch'*, *border: GridStyle = [illegible]*, *show_borders: DiBool | None = None*)

Bases: object

An widget which displays a given color.

**get_style**() → str

Compute the style for the swatch..

**class** euporie.core.widgets.forms.**Text**(*text: str = ''*, *style: str = 'class:input'*, *height: int = 1*, *min_height: int = 1*, *multiline: FilterOrBool = False*, *expand: FilterOrBool = True*, *width: int | None = None*, *completer: Completer | None = None*, *options: list[str] | Callable[[], list[str]] | None = None*, *show_borders: DiBool | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *validation: Callable[[str], bool] | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *placeholder: str | None = None*, *lexer: Lexer | None = None*, *input_processors: Sequence[Processor] | None = None*, *disabled: FilterOrBool = False*, *password: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *prompt: AnyFormattedText | None = None*)

Bases: object

A text input widget.

**border_style**() → str

Calculate the style to apply to the widget's border.

**property text: str**

Return the input's text value.

**class** euporie.core.widgets.forms.**ToggleButton**(*text: AnyFormattedText*, *on_click: Callable[[ToggleButton], None] | None = None*, *width: int | None = None*, *style: str | Callable[[], str] = 'class:input'*, *border: GridStyle | None = [illegible]*, *show_borders: DiBool | None = None*, *selected: bool = False*, *disabled: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*)

Bases: *ToggleableWidget*

A toggleable button widget.

**container: AnyContainer**

**disabled: *Filter***

**key_bindings: *KeyBindingsBase* | None**

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Focus on mouse down and toggle state on mouse up.

**on_click: *Event***

**property selected: bool**

Return the selection state of the toggle button.

**toggle**() → None

Toggle the selected state and trigger the "clicked" callback.

**class** euporie.core.widgets.forms.**ToggleButtons**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, disabled: FilterOrBool = False, vertical: FilterOrBool = False*)

Bases: *SelectableWidget*

A widget where an option is selected using mutually exclusive toggle-buttons.

**get_button_style**(*index: int*) → Callable[[], str]

Return the current button style.

**hover_rel**(*rel: int*) → None

Hover an index relative to the current hovered index.

**property index: int | None**

Return the first selected index.

**property indices: list[int]**

Return a list of the selected indices.

**key_bindings**() → *KeyBindingsBase*

Return key-bindings for the drop-down widget.

**load_container**() → AnyContainer

Load the widget's container.

**property mask: list[bool]**

Get mask of selected options.

**mouse_handler**(*i: int, mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

**select_rel**(*rel: int*) → None

Select an index relative to the current index.

**property style: str**

Return the widget's style.

**toggle_item**(*index: int*) → None

Toggle the selection status of the option at a given index.

**update_buttons**(*widget:* euporie.core.widgets.forms.SelectableWidget | *None* = *None*) → None

Set the toggle buttons' selection state when the selected index changes.

**property value: Any**

Return the selected value.

**property values: list[Any]**

Return a list of the selected values.

**class** euporie.core.widgets.forms.**ToggleableWidget**

Bases: object

Base class for toggleable widgets.

**container: AnyContainer**

**disabled: *Filter***

**key_bindings: *KeyBindingsBase* | None**

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Focus on mouse down and toggle state on mouse up.

**on_click: *Event***

**selected: bool**

**toggle**() → None

Toggle the selected state and trigger the "clicked" callback.

## euporie.core.widgets.inputs

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

## Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *get_app*() | Get the current active (running) Application. |
| *get_lexer_by_name*(_alias, **options) | Return an instance of a *Lexer* subclass that has *alias* in its aliases list. |
| *has_focus*(value) | Enable when this buffer has the focus. |
| *is_true*(value) | Test whether *value* is True. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *merge_key_bindings*(bindings) | Merge multiple Keybinding objects together. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *scrollable*(window) | Return a filter which indicates if a window is scrollable. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |

### euporie.core.widgets.inputs.add_cmd

euporie.core.widgets.inputs.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.widgets.inputs.add_setting

euporie.core.widgets.inputs.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.widgets.inputs.get_app

euporie.core.widgets.inputs.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.inputs.get_lexer_by_name

euporie.core.widgets.inputs.**get_lexer_by_name**(*_alias*, ***options*)

Return an instance of a *Lexer* subclass that has *alias* in its aliases list. The lexer is given the *options* at its instantiation.

Will raise *pygments.util.ClassNotFound* if no lexer with that alias is found.

### euporie.core.widgets.inputs.has_focus

euporie.core.widgets.inputs.**has_focus**(*value: FocusableElement*) → *Condition*

Enable when this buffer has the focus.

### euporie.core.widgets.inputs.is_true

euporie.core.widgets.inputs.**is_true**(*value: Union[Filter, bool]*) → bool

Test whether *value* is True. In case of a Filter, call it.

**Parameters**

**value** – Boolean or *Filter* instance.

### euporie.core.widgets.inputs.load_registered_bindings

euporie.core.widgets.inputs.**load_registered_bindings**(**names: str, config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.core.widgets.inputs.merge_key_bindings

euporie.core.widgets.inputs.**merge_key_bindings**(*bindings: Sequence[*KeyBindingsBase*]*) → _MergedKeyBindings

Merge multiple `Keybinding` objects together.

Usage:

```
bindings = merge_key_bindings([bindings1, bindings2, ...])
```

### euporie.core.widgets.inputs.register_bindings

euporie.core.widgets.inputs.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.widgets.inputs.scrollable

euporie.core.widgets.inputs.**scrollable**(*window:* Window) → *Filter*

Return a filter which indicates if a window is scrollable.

### euporie.core.widgets.inputs.to_filter

euporie.core.widgets.inputs.**to_filter**(*bool_or_filter: Union[*Filter, *bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### Classes

| | |
|---|---|
| *AppendLineAutoSuggestion*([style]) | Append the auto suggestion to the current line of the input. |
| *AutoSuggest*() | Base class for auto suggestion implementations. |
| *BeforeInput*(text[, style]) | Insert text before the input. |
| *Buffer*([completer, auto_suggest, history, ...]) | The core data structure that holds the text and cursor position of the current input line and implements all text manipulations on top of it. |
| *BufferControl*([buffer, input_processors, ...]) | Control for visualizing the content of a *Buffer*. |
| *Completer*() | Base class for completer implementations. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalAutoSuggestAsync*(auto_suggest, filter) | Auto suggest that can be turned on and of according to a certain condition. |

continues on next page

Table 10 – continued from previous page

| | |
|---|---|
| ConditionalContainer(content, filter) | Wrapper around any other container that can change the visibility. |
| ConditionalMargin(margin, filter) | Wrapper around other Margin classes to show/hide them. |
| ConditionalProcessor(processor, filter) | Processor that applies another processor, according to a certain condition. Example::. |
| D | alias of Dimension |
| DiagnosticProcessor(report[, style]) | Highlight diagnostics. |
| DisplayMultipleCursors() | When we're in Vi block insert mode, display all the cursors. |
| Document([text, cursor_position, selection]) | This is a immutable class around the text and cursor position, and contains methods for querying this data, e.g. |
| DynamicAutoSuggest(get_auto_suggest) | Validator class that can dynamically returns any Validator. |
| DynamicCompleter(get_completer) | Completer class that can dynamically returns any Completer. |
| DynamicLexer(get_lexer) | Lexer class that can dynamically returns any Lexer. |
| DynamicValidator(get_validator) | Validator class that can dynamically returns any Validator. |
| HighlightIncrementalSearchProcessor() | Highlight the search terms that are used for highlighting the incremental search. |
| HighlightMatchingBracketProcessor([chars, ...]) | When the cursor is on or right after a bracket, it highlights the matching bracket. |
| HighlightSelectionProcessor() | Processor that highlights the selection in the document. |
| KernelInput(kernel_tab[, text, multiline, ...]) | Kernel input text areas. |
| MarginContainer(margin, target) | A container which renders a stand-alone margin. |
| NumberedMargin([diagnostics, show_diagnostics]) | Margin that displays the line numbers of a Window. |
| OverflowMargin() | A margin which indicates lines extending beyond the edge of the window. |
| PagerState(code, cursor_pos, data) | A named tuple which describes the state of a pager. |
| PasswordProcessor([char]) | Processor that masks the input. |
| Processor() | Manipulate the fragments for a given line in a BufferControl. |
| PygmentsLexer(pygments_lexer_cls[, ...]) | Lexer that calls a pygments lexer. |
| Report([iterable]) | Class for storing a diagnostic report. |
| ScrollOffsets([top, bottom, left, right]) | Scroll offsets for the Window class. |
| ScrollbarMargin([display_arrows, ...]) | Margin displaying a scrollbar. |
| SearchToolbar([search_buffer, vi_mode, ...]) | **param vi_mode** Display '/' and '?' instead of I-search. |
| ShowTrailingWhiteSpaceProcessor([char, style]) | Make trailing whitespace visible. |
| SimpleLexer([style]) | Lexer that doesn't do any tokenizing and returns the whole input as one token. |
| StdInput(kernel_tab) | A widget to accept kernel input. |
| TabsProcessor([tabstop, char1, char2, style]) | Render tabs as spaces (instead of ^I) or make them visible (for instance, by replacing them with dots.) |
| TextArea([text, multiline, password, lexer, ...]) | A simple input field. |
| VSplit(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| Validator() | Abstract base class for an input validator. |
| Window([content, width, height, z_index, ...]) | Container that holds a control. |

### euporie.core.widgets.inputs.AppendLineAutoSuggestion

**class** euporie.core.widgets.inputs.**AppendLineAutoSuggestion**(*style: str = 'class:auto-suggestion'*)

Append the auto suggestion to the current line of the input.

### euporie.core.widgets.inputs.AutoSuggest

**class** euporie.core.widgets.inputs.**AutoSuggest**

Base class for auto suggestion implementations.

### euporie.core.widgets.inputs.BeforeInput

**class** euporie.core.widgets.inputs.**BeforeInput**(*text: AnyFormattedText*, *style: str = ''*)

Insert text before the input.

**Parameters**

- **text** – This can be either plain text or formatted text (or a callable that returns any of those).
- **style** – style to be applied to this prompt/prefix.

### euporie.core.widgets.inputs.Buffer

**class** euporie.core.widgets.inputs.**Buffer**(*completer: prompt_toolkit.completion.base.Completer | None = None*, *auto_suggest: prompt_toolkit.auto_suggest.AutoSuggest | None = None*, *history: prompt_toolkit.history.History | None = None*, *validator: prompt_toolkit.validation.Validator | None = None*, *tempfile_suffix: Union[str, Callable[[], str]] = ''*, *tempfile: Union[str, Callable[[], str]] = ''*, *name: str = ''*, *complete_while_typing: Union[Filter, bool] = False*, *validate_while_typing: Union[Filter, bool] = False*, *enable_history_search: Union[Filter, bool] = False*, *document: prompt_toolkit.document.Document | None = None*, *accept_handler: Optional[Callable[[Buffer], bool]] = None*, *read_only: Union[Filter, bool] = False*, *multiline: Union[Filter, bool] = True*, *on_text_changed: Optional[Callable[[Buffer], None]] = None*, *on_text_insert: Optional[Callable[[Buffer], None]] = None*, *on_cursor_position_changed: Optional[Callable[[Buffer], None]] = None*, *on_completions_changed: Optional[Callable[[Buffer], None]] = None*, *on_suggestion_set: Optional[Callable[[Buffer], None]] = None*)

The core data structure that holds the text and cursor position of the current input line and implements all text manipulations on top of it. It also implements the history, undo stack and the completion state.

**Parameters**

- **completer** – Completer instance.

- **history** – *History* instance.
- **tempfile_suffix** – The tempfile suffix (extension) to be used for the "open in editor" function. For a Python REPL, this would be ".py", so that the editor knows the syntax highlighting to use. This can also be a callable that returns a string.
- **tempfile** – For more advanced tempfile situations where you need control over the subdirectories and filename. For a Git Commit Message, this would be ".git/COMMIT_EDITMSG", so that the editor knows the syntax highlighting to use. This can also be a callable that returns a string.
- **name** – Name for this buffer. E.g. DEFAULT_BUFFER. This is mostly useful for key bindings where we sometimes prefer to refer to a buffer by their name instead of by reference.
- **accept_handler** – Called when the buffer input is accepted. (Usually when the user presses *enter*.) The accept handler receives this *Buffer* as input and should return True when the buffer text should be kept instead of calling reset.

  In case of a *PromptSession* for instance, we want to keep the text, because we will exit the application, and only reset it during the next run.

Events:

**Parameters**

- **on_text_changed** – When the buffer text changes. (Callable or None.)
- **on_text_insert** – When new text is inserted. (Callable or None.)
- **on_cursor_position_changed** – When the cursor moves. (Callable or None.)
- **on_completions_changed** – When the completions were changed. (Callable or None.)
- **on_suggestion_set** – When an auto-suggestion text has been set. (Callable or None.)

Filters:

**Parameters**

- **complete_while_typing** – Filter or *bool*. Decide whether or not to do asynchronous autocompleting while typing.
- **validate_while_typing** – Filter or *bool*. Decide whether or not to do asynchronous validation while typing.
- **enable_history_search** – Filter or *bool* to indicate when up-arrow partial string matching is enabled. It is advised to not enable this at the same time as *complete_while_typing*, because when there is an autocompletion found, the up arrows usually browse through the completions, rather than through the history.
- **read_only** – Filter. When True, changes will not be allowed.
- **multiline** – Filter or *bool*. When not set, pressing *Enter* will call the *accept_handler*. Otherwise, pressing *Esc-Enter* is required.

### euporie.core.widgets.inputs.BufferControl

**class** euporie.core.widgets.inputs.**BufferControl**(*buffer:* Buffer | *None* = *None*, *input_processors: list[*Processor*]* | *None* = *None*, *include_default_input_processors:* bool = *True*, *lexer:* Lexer | *None* = *None*, *preview_search:* FilterOrBool = *False*, *focusable:* FilterOrBool = *True*, *search_buffer_control:* *None* | SearchBufferControl | *Callable[[],* SearchBufferControl*]* = *None*, *menu_position: Callable[[],* int | *None]* | *None* = *None*, *focus_on_click:* FilterOrBool = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*)

Control for visualizing the content of a *Buffer*.

**Parameters**

- **buffer** – The *Buffer* object to be displayed.
- **input_processors** – A list of *Processor* objects.
- **include_default_input_processors** – When True, include the default processors for highlighting of selection, search and displaying of multiple cursors.
- **lexer** – Lexer instance for syntax highlighting.
- **preview_search** – *bool* or *Filter*: Show search while typing. When this is *True*, probably you want to add a HighlightIncrementalSearchProcessor as well. Otherwise only the cursor position will move, but the text won't be highlighted.
- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **focus_on_click** – Focus this buffer when it's click, but not yet focused.
- **key_bindings** – a *KeyBindings* object.

### euporie.core.widgets.inputs.Completer

**class** euporie.core.widgets.inputs.**Completer**

Base class for completer implementations.

### euporie.core.widgets.inputs.Condition

**class** euporie.core.widgets.inputs.**Condition**(*func: Callable[[],* bool*]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.inputs.ConditionalAutoSuggestAsync

**class** euporie.core.widgets.inputs.**ConditionalAutoSuggestAsync**(*auto_suggest:* AutoSuggest, *filter:* *bool* | Filter)

Auto suggest that can be turned on and of according to a certain condition.

### euporie.core.widgets.inputs.ConditionalContainer

**class** euporie.core.widgets.inputs.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.inputs.ConditionalMargin

**class** euporie.core.widgets.inputs.**ConditionalMargin**(*margin:* Margin, *filter:* *Union*[Filter, *bool*])

Wrapper around other *Margin* classes to show/hide them.

### euporie.core.widgets.inputs.ConditionalProcessor

**class** euporie.core.widgets.inputs.**ConditionalProcessor**(*processor:* Processor, *filter:* *Union*[Filter, *bool*])

Processor that applies another processor, according to a certain condition. Example:

```
# Create a function that returns whether or not the processor should
# currently be applied.
def highlight_enabled():
    return true_or_false

# Wrapped it in a `ConditionalProcessor` for usage in a `BufferControl`.
BufferControl(input_processors=[
    ConditionalProcessor(HighlightSearchProcessor(),
                         Condition(highlight_enabled))])
```

**Parameters**

- **processor** – *Processor* instance.
- **filter** – Filter instance.

### euporie.core.widgets.inputs.D

euporie.core.widgets.inputs.**D**

alias of *Dimension*

### euporie.core.widgets.inputs.DiagnosticProcessor

**class** euporie.core.widgets.inputs.**DiagnosticProcessor**(*report:* Report | *Callable[[],* Report*]*, *style:* *str* = *'underline'*)

Highlight diagnostics.

### euporie.core.widgets.inputs.DisplayMultipleCursors

**class** euporie.core.widgets.inputs.**DisplayMultipleCursors**

When we're in Vi block insert mode, display all the cursors.

### euporie.core.widgets.inputs.Document

**class** euporie.core.widgets.inputs.**Document**(*text:* *str* = *''*, *cursor_position:* *int* | *None* = *None*, *selection:* prompt_toolkit.selection.SelectionState | *None* = *None*)

This is a immutable class around the text and cursor position, and contains methods for querying this data, e.g. to give the text before the cursor.

This class is usually instantiated by a *Buffer* object, and accessed as the *document* property of that class.

**Parameters**

- **text** – string
- **cursor_position** – int
- **selection** – *SelectionState*

### euporie.core.widgets.inputs.DynamicAutoSuggest

**class** euporie.core.widgets.inputs.**DynamicAutoSuggest**(*get_auto_suggest: Callable[[],* prompt_toolkit.auto_suggest.AutoSuggest | *None]*)

Validator class that can dynamically returns any Validator.

**Parameters**

**get_validator** – Callable that returns a *Validator* instance.

### euporie.core.widgets.inputs.DynamicCompleter

**class** euporie.core.widgets.inputs.**DynamicCompleter**(*get_completer: Callable[[], prompt_toolkit.completion.base.Completer | None]*)

Completer class that can dynamically returns any Completer.

**Parameters**
**get_completer** – Callable that returns a *Completer* instance.

### euporie.core.widgets.inputs.DynamicLexer

**class** euporie.core.widgets.inputs.**DynamicLexer**(*get_lexer: Callable[[], prompt_toolkit.lexers.base.Lexer | None]*)

Lexer class that can dynamically returns any Lexer.

**Parameters**
**get_lexer** – Callable that returns a `Lexer` instance.

### euporie.core.widgets.inputs.DynamicValidator

**class** euporie.core.widgets.inputs.**DynamicValidator**(*get_validator: Callable[[], prompt_toolkit.validation.Validator | None]*)

Validator class that can dynamically returns any Validator.

**Parameters**
**get_validator** – Callable that returns a *Validator* instance.

### euporie.core.widgets.inputs.HighlightIncrementalSearchProcessor

**class** euporie.core.widgets.inputs.**HighlightIncrementalSearchProcessor**

Highlight the search terms that are used for highlighting the incremental search. The style class 'incsearch' will be applied to the content.

Important: this requires the *preview_search=True* flag to be set for the *BufferControl*. Otherwise, the cursor position won't be set to the search match while searching, and nothing happens.

### euporie.core.widgets.inputs.HighlightMatchingBracketProcessor

**class** euporie.core.widgets.inputs.**HighlightMatchingBracketProcessor**(*chars: str = '[](){}<>', max_cursor_distance: int = 1000*)

When the cursor is on or right after a bracket, it highlights the matching bracket.

**Parameters**

**max_cursor_distance** – Only highlight matching brackets when the cursor is within this distance. (From inside a *Processor*, we can't know which lines will be visible on the screen. But we also don't want to scan the whole document for matching brackets on each key press, so we limit to this value.)

### euporie.core.widgets.inputs.HighlightSelectionProcessor

**class** euporie.core.widgets.inputs.**HighlightSelectionProcessor**

Processor that highlights the selection in the document.

### euporie.core.widgets.inputs.KernelInput

**class** euporie.core.widgets.inputs.**KernelInput**(*kernel_tab: KernelTab*, *text: str = ''*, *multiline: FilterOrBool = True*, *password: FilterOrBool = False*, *lexer: Lexer | None = None*, *auto_suggest: AutoSuggest | None = None*, *completer: Completer | None = None*, *complete_while_typing: FilterOrBool = True*, *validator: Validator | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *history: History | None = None*, *focusable: FilterOrBool = True*, *focus_on_click: FilterOrBool = True*, *wrap_lines: FilterOrBool = False*, *read_only: FilterOrBool = False*, *width: AnyDimension = None*, *height: AnyDimension = None*, *dont_extend_height: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *line_numbers: bool = False*, *get_line_prefix: GetLinePrefixCallable | None = None*, *scrollbar: FilterOrBool = True*, *style: str = 'class:kernel-input'*, *search_field: SearchToolbar | None = None*, *preview_search: FilterOrBool = False*, *prompt: AnyFormattedText = ''*, *input_processors: list[Processor] | None = None*, *name: str = ''*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *on_cursor_position_changed: Callable[[Buffer], None] | None = None*, *tempfile_suffix: str | Callable[[], str] = ''*, *key_bindings: KeyBindingsBase | None = None*, *enable_history_search: FilterOrBool = False*, *autosuggest_while_typing: FilterOrBool = True*, *validate_while_typing: FilterOrBool = False*, *scroll_offsets: ScrollOffsets | None = None*, *formatters: list[Formatter] | None = None*, *language: str | Callable[[], str] | None = None*, *diagnostics: Report | Callable[[], Report] | None = None*, *inspector: Inspector | None = None*, *show_diagnostics: FilterOrBool = True*)

Kernel input text areas.

A customized text area for the cell input.

### euporie.core.widgets.inputs.MarginContainer

**class** euporie.core.widgets.inputs.**MarginContainer**(*margin:* Margin, *target:* *ScrollableContainer*)

A container which renders a stand-alone margin.

### euporie.core.widgets.inputs.NumberedMargin

**class** euporie.core.widgets.inputs.**NumberedMargin**(*diagnostics:* Report | *Callable[[],* Report*]* | *None* = *None*, *show_diagnostics:* *FilterOrBool* = *False*)

Margin that displays the line numbers of a *Window*.

### euporie.core.widgets.inputs.OverflowMargin

**class** euporie.core.widgets.inputs.**OverflowMargin**

A margin which indicates lines extending beyond the edge of the window.

### euporie.core.widgets.inputs.PagerState

**class** euporie.core.widgets.inputs.**PagerState**(*code: str*, *cursor_pos: int*, *data: dict*)

A named tuple which describes the state of a pager.

### euporie.core.widgets.inputs.PasswordProcessor

**class** euporie.core.widgets.inputs.**PasswordProcessor**(*char: str* = *'*'*)

Processor that masks the input. (For passwords.)

**Parameters**

**char** – (string) Character to be used. "*" by default.

### euporie.core.widgets.inputs.Processor

**class** euporie.core.widgets.inputs.**Processor**

Manipulate the fragments for a given line in a *BufferControl*.

### euporie.core.widgets.inputs.PygmentsLexer

**class** euporie.core.widgets.inputs.**PygmentsLexer**(*pygments_lexer_cls: type[PygmentsLexerCls]*, *sync_from_start: FilterOrBool = True*, *syntax_sync: SyntaxSync | None = None*)

Lexer that calls a pygments lexer.

Example:

```
from pygments.lexers.html import HtmlLexer
lexer = PygmentsLexer(HtmlLexer)
```

Note: Don't forget to also load a Pygments compatible style. E.g.:

```
from prompt_toolkit.styles.from_pygments import style_from_pygments_cls
from pygments.styles import get_style_by_name
style = style_from_pygments_cls(get_style_by_name('monokai'))
```

**Parameters**

- **pygments_lexer_cls** – A *Lexer* from Pygments.
- **sync_from_start** – Start lexing at the start of the document. This will always give the best results, but it will be slow for bigger documents. (When the last part of the document is display, then the whole document will be lexed by Pygments on every key stroke.) It is recommended to disable this for inputs that are expected to be more than 1,000 lines.
- **syntax_sync** – *SyntaxSync* object.

### euporie.core.widgets.inputs.Report

**class** euporie.core.widgets.inputs.**Report**(*iterable=()*, */*)

Class for storing a diagnostic report.

### euporie.core.widgets.inputs.ScrollOffsets

**class** euporie.core.widgets.inputs.**ScrollOffsets**(*top: Union[int, Callable[[], int]] = 0*, *bottom: Union[int, Callable[[], int]] = 0*, *left: Union[int, Callable[[], int]] = 0*, *right: Union[int, Callable[[], int]] = 0*)

Scroll offsets for the *Window* class.

Note that left/right offsets only make sense if line wrapping is disabled.

### euporie.core.widgets.inputs.ScrollbarMargin

**class** euporie.core.widgets.inputs.**ScrollbarMargin**(*display_arrows: Union[Filter, bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[Filter, bool] = False*, *smooth: bool = True*, *style: str = ''*)

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

### euporie.core.widgets.inputs.SearchToolbar

**class** euporie.core.widgets.inputs.**SearchToolbar**(*search_buffer: Buffer | None = None*, *vi_mode: bool = False*, *text_if_not_searching: AnyFormattedText = ''*, *forward_search_prompt: AnyFormattedText = 'I-search: '*, *backward_search_prompt: AnyFormattedText = 'I-search backward: '*, *ignore_case: FilterOrBool = False*)

**Parameters**

- **vi_mode** – Display '/' and '?' instead of I-search.
- **ignore_case** – Search case insensitive.

### euporie.core.widgets.inputs.ShowTrailingWhiteSpaceProcessor

**class** euporie.core.widgets.inputs.**ShowTrailingWhiteSpaceProcessor**(*char: str = '·'*, *style: str = 'class:trailing-whitespace'*)

Make trailing whitespace visible.

### euporie.core.widgets.inputs.SimpleLexer

**class** euporie.core.widgets.inputs.**SimpleLexer**(*style: str = ''*)

Lexer that doesn't do any tokenizing and returns the whole input as one token.

**Parameters**

**style** – The style string for this lexer.

### euporie.core.widgets.inputs.StdInput

**class** euporie.core.widgets.inputs.**StdInput**(*kernel_tab:* KernelTab)

A widget to accept kernel input.

### euporie.core.widgets.inputs.TabsProcessor

**class** euporie.core.widgets.inputs.**TabsProcessor**(*tabstop: Union[int, Callable[[], int]] = 4*, *char1: Union[str, Callable[[], str]] = '\'*, *char2: Union[str, Callable[[], str]] = '␣'*, *style: str = 'class:tab'*)

Render tabs as spaces (instead of ^I) or make them visible (for instance, by replacing them with dots.)

**Parameters**

- **tabstop** – Horizontal space taken by a tab. (*int* or callable that returns an *int*).
- **char1** – Character or callable that returns a character (text of length one). This one is used for the first space taken by the tab.
- **char2** – Like *char1*, but for the rest of the space.

### euporie.core.widgets.inputs.TextArea

**class** euporie.core.widgets.inputs.**TextArea**(*text: str = ''*, *multiline: FilterOrBool = True*, *password: FilterOrBool = False*, *lexer: Lexer | None = None*, *auto_suggest:* AutoSuggest *| None = None*, *completer:* Completer *| None = None*, *complete_while_typing: FilterOrBool = True*, *validator:* Validator *| None = None*, *accept_handler: BufferAcceptHandler | None = None*, *history:* History *| None = None*, *focusable: FilterOrBool = True*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = True*, *read_only: FilterOrBool = False*, *width: AnyDimension = None*, *height: AnyDimension = None*, *dont_extend_height: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *line_numbers: bool = False*, *get_line_prefix: GetLinePrefixCallable | None = None*, *scrollbar: bool = False*, *style: str = ''*, *search_field:* SearchToolbar *| None = None*, *preview_search: FilterOrBool = True*, *prompt: AnyFormattedText = ''*, *input_processors: list[*Processor*] | None = None*, *name: str = ''*)

A simple input field.

This is a higher level abstraction on top of several other classes with sane defaults.

This widget does have the most common options, but it does not intend to cover every single use case. For more configurations options, you can always build a text area manually, using a `Buffer`, `BufferControl` and `Window`.

Buffer attributes:

**Parameters**

- **text** – The initial text.
- **multiline** – If True, allow multiline input.
- **completer** – `Completer` instance for auto completion.
- **complete_while_typing** – Boolean.
- **accept_handler** – Called when *Enter* is pressed (This should be a callable that takes a buffer as input).
- **history** – *`History`* instance.
- **auto_suggest** – *`AutoSuggest`* instance for input suggestions.

BufferControl attributes:

**Parameters**

- **password** – When *True*, display using asterisks.
- **focusable** – When *True*, allow this widget to receive the focus.
- **focus_on_click** – When *True*, focus after mouse click.
- **input_processors** – *None* or a list of `Processor` objects.
- **validator** – *None* or a *`Validator`* object.

Window attributes:

**Parameters**

- **lexer** – `Lexer` instance for syntax highlighting.
- **wrap_lines** – When *True*, don't scroll horizontally, but wrap lines.
- **width** – Window width. (`Dimension` object.)
- **height** – Window height. (`Dimension` object.)
- **scrollbar** – When *True*, display a scroll bar.
- **style** – A style string.
- **dont_extend_width** – When *True*, don't take up more width then the preferred width reported by the control.
- **dont_extend_height** – When *True*, don't take up more width then the preferred height reported by the control.
- **get_line_prefix** – None or a callable that returns formatted text to be inserted before a line. It takes a line number (int) and a wrap_count and returns formatted text. This can be used for implementation of line continuations, things like Vim "breakindent" and so on.

Other attributes:

**Parameters**

**search_field** – An optional *SearchToolbar* object.

### euporie.core.widgets.inputs.VSplit

**class** euporie.core.widgets.inputs.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: HorizontalAlign* = *HorizontalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.inputs.Validator

**class** euporie.core.widgets.inputs.**Validator**

Abstract base class for an input validator.

A validator is typically created in one of the following two ways:

- Either by overriding this class and implementing the *validate* method.
- Or by passing a callable to *Validator.from_callable*.

If the validation takes some time and needs to happen in a background thread, this can be wrapped in a `ThreadedValidator`.

### euporie.core.widgets.inputs.Window

**class** euporie.core.widgets.inputs.**Window**(*content:* UIControl | *None* = *None*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *dont_extend_width: FilterOrBool* = *False*, *dont_extend_height: FilterOrBool* = *False*, *ignore_content_width: FilterOrBool* = *False*, *ignore_content_height: FilterOrBool* = *False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool* = *False*, *wrap_lines: FilterOrBool* = *False*, *get_vertical_scroll: Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool* = *False*, *cursorline: FilterOrBool* = *False*, *cursorcolumn: FilterOrBool* = *False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]]* = *None*, *align:* WindowAlign | *Callable[[],* WindowAlign*]* = *WindowAlign.LEFT*, *style: str* | *Callable[[], str]* = *''*, *char: None* | *str* | *Callable[[], str]* = *None*, *get_line_prefix: GetLinePrefixCallable* | *None* = *None*)

Container that holds a control.

### Exceptions

| | |
|---|---|
| *ClassNotFound* | Raised if one of the lookup functions didn't find a matching class. |

### euporie.core.widgets.inputs.ClassNotFound

**exception** euporie.core.widgets.inputs.**ClassNotFound**

Raised if one of the lookup functions didn't find a matching class.

**class** euporie.core.widgets.inputs.**KernelInput**(*kernel_tab: KernelTab*, *text: str = ''*, *multiline: FilterOrBool = True*, *password: FilterOrBool = False*, *lexer: Lexer | None = None*, *auto_suggest: AutoSuggest | None = None*, *completer: Completer | None = None*, *complete_while_typing: FilterOrBool = True*, *validator: Validator | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *history: History | None = None*, *focusable: FilterOrBool = True*, *focus_on_click: FilterOrBool = True*, *wrap_lines: FilterOrBool = False*, *read_only: FilterOrBool = False*, *width: AnyDimension = None*, *height: AnyDimension = None*, *dont_extend_height: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *line_numbers: bool = False*, *get_line_prefix: GetLinePrefixCallable | None = None*, *scrollbar: FilterOrBool = True*, *style: str = 'class:kernel-input'*, *search_field: SearchToolbar | None = None*, *preview_search: FilterOrBool = False*, *prompt: AnyFormattedText = ''*, *input_processors: list[Processor] | None = None*, *name: str = ''*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *on_cursor_position_changed: Callable[[Buffer], None] | None = None*, *tempfile_suffix: str | Callable[[], str] = ''*, *key_bindings: KeyBindingsBase | None = None*, *enable_history_search: FilterOrBool = False*, *autosuggest_while_typing: FilterOrBool = True*, *validate_while_typing: FilterOrBool = False*, *scroll_offsets: ScrollOffsets | None = None*, *formatters: list[Formatter] | None = None*, *language: str | Callable[[], str] | None = None*, *diagnostics: Report | Callable[[], Report] | None = None*, *inspector: Inspector | None = None*, *show_diagnostics: FilterOrBool = True*)

Bases: *TextArea*

Kernel input text areas.

A customized text area for the cell input.

**property accept_handler: Optional[Callable[[*Buffer*], bool]]**

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

**property current_diagnostic_message: StyleAndTextTuples**

Format the currently selected diagnostic message.

**property diagnostics: *Report***

The current diagnostics report.

**property document: *Document***

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

**async inspect**(*auto: bool = False*) → None

Get contextual help for the current cursor position in the current cell.

**property language: str**

The current language of the text in the input box.

**reformat**() → None

Reformat the cell's input.

**property text: str**

The *Buffer* text.

**class** euporie.core.widgets.inputs.**StdInput**(*kernel_tab:* KernelTab)

Bases: object

A widget to accept kernel input.

**accept**(*buffer:* Buffer) → bool

Send the input to the kernel and hide the input box.

**get_input**(*prompt: str = 'Please enter a value: '*, *password: bool = False*) → None

Prompt the user for input and sends the result to the kernel.

## euporie.core.widgets.layout

Define widget for defining layouts.

### Functions

| | |
|---|---|
| *NamedTuple*(typename[, fields]) | Typed version of namedtuple. |
| *abstractmethod*(funcobj) | A decorator indicating abstract methods. |
| *cast*(typ, val) | Cast a value to a type. |
| *fragment_list_width*(fragments) | Return the character width of this text fragment list. |
| *get_app*() | Get the current active (running) Application. |
| *to_container*(container) | Monkey-patch *to_container* to collect container status functions. |
| *to_dimension*(value) | Turn the given object into a *Dimension* object. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |
| *truncate*(ft, width[, style, placeholder, ...]) | Truncate all lines at a given length. |

### euporie.core.widgets.layout.NamedTuple

euporie.core.widgets.layout.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)

Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

### euporie.core.widgets.layout.abstractmethod

euporie.core.widgets.layout.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is ABCMeta or derived from it. A class that has a metaclass derived from ABCMeta cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Usage:

**class C(metaclass=ABCMeta):**
@abstractmethod def my_abstract_method(self, arg1, arg2, argN):

…

### euporie.core.widgets.layout.cast

euporie.core.widgets.layout.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.core.widgets.layout.fragment_list_width

euporie.core.widgets.layout.**fragment_list_width**(*fragments: StyleAndTextTuples*) → int

Return the character width of this text fragment list. (Take double width characters into account.)

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.layout.get_app

euporie.core.widgets.layout.**get_app**() → *Application*[Any]

Get the current active (running) Application. An *Application* is active during the `Application.run_async()` call.

We assume that there can only be one *Application* active at the same time. There is only one terminal window, with only one stdin and stdout. This makes the code significantly easier than passing around the *Application* everywhere.

If no *Application* is running, then return by default a `DummyApplication`. For practical reasons, we prefer to not raise an exception. This way, we don't have to check all over the place whether an actual *Application* was returned.

(For applications like pymux where we can have more than one *Application*, we'll use a work-around to handle that.)

### euporie.core.widgets.layout.to_container

euporie.core.widgets.layout.**to_container**(*container: AnyContainer*) → *Container*

Monkey-patch *to_container* to collect container status functions.

### euporie.core.widgets.layout.to_dimension

euporie.core.widgets.layout.**to_dimension**(*value: Union[None, int,* Dimension*, Callable[[], Any]]*) → *Dimension*

Turn the given object into a *Dimension* object.

### euporie.core.widgets.layout.to_filter

euporie.core.widgets.layout.**to_filter**(*bool_or_filter: Union[*Filter*, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### euporie.core.widgets.layout.to_formatted_text

euporie.core.widgets.layout.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

### euporie.core.widgets.layout.truncate

euporie.core.widgets.layout.**truncate**(*ft: StyleAndTextTuples*, *width: int*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Truncate all lines at a given length.

**Parameters**

- **ft** – The formatted text to truncate
- **width** – The width at which to truncate the text
- **style** – The style to apply to the truncation placeholder. The style of the truncated text will be used if not provided
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – Do not use placeholder when truncating whitespace

**Returns**

The truncated formatted text

## Classes

| | |
|---|---|
| *ABCMeta*(name, bases, namespace, /, **kwargs) | Metaclass for defining Abstract Base Classes (ABCs). |
| *AccordionSplit*(children, titles[, active, ...]) | A container which switches between children using expandable sections. |
| *Border*(body, border, style, str] =, show_borders) | Draw a border around any container. |
| *Box*(body[, padding, padding_left, ...]) | Add padding around a container. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *ConditionalSplit*(vertical, *args, **kwargs) | A split container where the orientation depends on a filter. |
| *D* | alias of *Dimension* |
| *DiBool*([top, right, bottom, left]) | A tuple of four bools with directions. |
| *DynamicContainer*(get_container) | Container class that dynamically returns any Container. |
| *Event*(sender[, handler]) | Simple event to which event handlers can be attached. For instance::. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *MouseButton*(value[, names, module, ...]) | |
| *MouseEventType*(value[, names, module, ...]) | |
| *ReferencedSplit*(split, children, *args, **kwargs) | A split container which maintains a reference to it's children. |
| *SimpleCache*([maxsize]) | Very simple cache that discards the oldest item when the cache size is exceeded. |
| *StackedSplit*(children, titles[, active, ...]) | Base class for containers with selectable children. |
| *TabBarControl*(tabs, active[, spacing, ...]) | A control which shows a tab bar. |
| *TabBarTab*(title, on_activate[, ...]) | A named tuple represting a tab and it's callbacks. |
| *TabbedSplit*(children, titles, active, style, ...) | A container which switches between children using tabs. |
| *UIContent*(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| *UIControl*() | Base class for all user interface controls. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.widgets.layout.ABCMeta

**class** euporie.core.widgets.layout.**ABCMeta**(*name*, *bases*, *namespace*, */*, ***kwargs*)

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

### euporie.core.widgets.layout.AccordionSplit

**class** euporie.core.widgets.layout.**AccordionSplit**(*children: Sequence[AnyContainer]*, *titles: Sequence[AnyFormattedText]*, *active: int = 0*, *style: str | Callable[[], str] = 'class:tab-split'*, *on_change: Callable[[StackedSplit], None] | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*)

A container which switches between children using expandable sections.

### euporie.core.widgets.layout.Border

**class** euporie.core.widgets.layout.**Border**(*body: AnyContainer*, *border: GridStyle | None = □□□□ □ □□ □□□□ □□□□*, *style: str | Callable[[], str] = 'class:border'*, *show_borders: DiBool | None = None*)

Draw a border around any container.

### euporie.core.widgets.layout.Box

**class** euporie.core.widgets.layout.**Box**(*body: AnyContainer*, *padding: AnyDimension = None*, *padding_left: AnyDimension = None*, *padding_right: AnyDimension = None*, *padding_top: AnyDimension = None*, *padding_bottom: AnyDimension = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *char: None | str | Callable[[], str] = None*, *modal: bool = False*, *key_bindings: KeyBindings | None = None*)

Add padding around a container.

This also makes sure that the parent can provide more space than required by the child. This is very useful when wrapping a small element with a fixed size into a `VSplit` or `HSplit` object. The `HSplit` and `VSplit` try to make sure to adapt respectively the width and height, possibly shrinking other elements. Wrapping something in a `Box` makes it flexible.

**Parameters**

- **body** – Another container object.
- **padding** – The margin to be used around the body. This can be
- **padding_left** (*overridden by*) – *padding_bottom*.
- **padding_right`** – *padding_bottom*.
- **and** (*padding_top*) – *padding_bottom*.
- **style** – A style string.
- **char** – Character to be used for filling the space around the body. (This is supposed to be a character with a terminal width of 1.)

### euporie.core.widgets.layout.Condition

**class** euporie.core.widgets.layout.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.layout.ConditionalContainer

**class** euporie.core.widgets.layout.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.layout.ConditionalSplit

**class** euporie.core.widgets.layout.**ConditionalSplit**(*vertical: FilterOrBool*, **args: Any*, ***kwargs: Any*)

A split container where the orientation depends on a filter.

### euporie.core.widgets.layout.D

euporie.core.widgets.layout.**D**

alias of *Dimension*

### euporie.core.widgets.layout.DiBool

**class** euporie.core.widgets.layout.**DiBool**(*top: bool = False*, *right: bool = False*, *bottom: bool = False*, *left: bool = False*)

A tuple of four bools with directions.

### euporie.core.widgets.layout.DynamicContainer

**class** euporie.core.widgets.layout.**DynamicContainer**(*get_container: Callable[[], AnyContainer]*)

Container class that dynamically returns any Container.

**Parameters**

**get_container** – Callable that returns a *Container* instance or any widget with a `__pt_container__` method.

### euporie.core.widgets.layout.Event

**class** euporie.core.widgets.layout.**Event**(*sender: _Sender*, *handler: Optional[Callable[[_Sender], None]] = None*)

Simple event to which event handlers can be attached. For instance:

```
class Cls:
    def __init__(self):
        # Define event. The first parameter is the sender.
        self.event = Event(self)

obj = Cls()

def handler(sender):
    pass

# Add event handler by using the += operator.
obj.event += handler

# Fire event.
obj.event()
```

### euporie.core.widgets.layout.FormattedTextControl

**class** euporie.core.widgets.layout.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.

- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **`focusable`** – *bool* or `Filter`: Tell whether this control is focusable.
- **`text`** – Text or formatted text to be displayed.
- **`style`** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **`key_bindings`** – a `KeyBindings` object.
- **`get_cursor_position`** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.widgets.layout.HSplit

**class** euporie.core.widgets.layout.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: VerticalAlign* = *VerticalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.layout.MouseButton

**class** euporie.core.widgets.layout.**MouseButton**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.layout.MouseEventType

**class** euporie.core.widgets.layout.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.layout.ReferencedSplit

**class** euporie.core.widgets.layout.**ReferencedSplit**(*split: type[_Split]*, *children: Sequence[AnyContainer]*, **args: Any*, ***kwargs: Any*)

A split container which maintains a reference to it's children.

### euporie.core.widgets.layout.SimpleCache

**class** euporie.core.widgets.layout.**SimpleCache**(*maxsize: int = 8*)

Very simple cache that discards the oldest item when the cache size is exceeded.

**Parameters**

**maxsize** – Maximum size of the cache. (Don't make it too big.)

### euporie.core.widgets.layout.StackedSplit

**class** euporie.core.widgets.layout.**StackedSplit**(*children: Sequence[AnyContainer]*, *titles: Sequence[AnyFormattedText]*, *active: int = 0*, *style: str | Callable[[], str] = 'class:tab-split'*, *on_change: Callable[[StackedSplit], None] | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*)

Base class for containers with selectable children.

### euporie.core.widgets.layout.TabBarControl

**class** euporie.core.widgets.layout.**TabBarControl**(*tabs: Sequence[TabBarTab] | Callable[[], Sequence[TabBarTab]]*, *active: int | Callable[[], int]*, *spacing: int = 1*, *closeable: bool = False*, *max_title_width: int = 30*)

A control which shows a tab bar.

### euporie.core.widgets.layout.TabBarTab

**class** euporie.core.widgets.layout.**TabBarTab**(*title: AnyFormattedText*, *on_activate: Callable*, *on_deactivate: Callable | None = None*, *on_close: Callable | None = None*)

A named tuple represting a tab and it's callbacks.

### euporie.core.widgets.layout.TabbedSplit

**class** euporie.core.widgets.layout.**TabbedSplit**(*children: Sequence[AnyContainer], titles: Sequence[AnyFormattedText], active: int = 0, style: str | Callable[[], str] = 'class:tab-split', on_change: Callable[[StackedSplit], None] | None = None, width: AnyDimension = None, height: AnyDimension = None, border: GridStyle = □□□□ □ □□ □□□□ □□□□, show_borders: DiBool | None = None*)

A container which switches between children using tabs.

### euporie.core.widgets.layout.UIContent

**class** euporie.core.widgets.layout.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>, line_count: int = 0, cursor_position: Point | None = None, menu_position: Point | None = None, show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.widgets.layout.UIControl

**class** euporie.core.widgets.layout.**UIControl**

Base class for all user interface controls.

### euporie.core.widgets.layout.VSplit

**class** euporie.core.widgets.layout.**VSplit**(*children: Sequence[AnyContainer], window_too_small: Container | None = None, align: HorizontalAlign = HorizontalAlign.JUSTIFY, padding: AnyDimension = 0, padding_char: str | None = None, padding_style: str = '', width: AnyDimension = None, height: AnyDimension = None, z_index: int | None = None, modal: bool = False, key_bindings: KeyBindingsBase | None = None, style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.layout.Window

**class** euporie.core.widgets.layout.**Window**(*content:* UIControl | *None* = *None*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *dont_extend_width: FilterOrBool* = *False*, *dont_extend_height: FilterOrBool* = *False*, *ignore_content_width: FilterOrBool* = *False*, *ignore_content_height: FilterOrBool* = *False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool* = *False*, *wrap_lines: FilterOrBool* = *False*, *get_vertical_scroll: Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool* = *False*, *cursorline: FilterOrBool* = *False*, *cursorcolumn: FilterOrBool* = *False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]]* = *None*, *align:* WindowAlign | *Callable[[],* WindowAlign*]* = *WindowAlign.LEFT*, *style: str* | *Callable[[], str]* = *''*, *char: None* | *str* | *Callable[[], str]* = *None*, *get_line_prefix: GetLinePrefixCallable* | *None* = *None*)

Container that holds a control.

### euporie.core.widgets.layout.partial

**class** euporie.core.widgets.layout.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.widgets.layout.**AccordionSplit**(*children: Sequence[AnyContainer]*, *titles: Sequence[AnyFormattedText]*, *active: int* = *0*, *style: str* | *Callable[[], str]* = *'class:tab-split'*, *on_change: Callable[[*StackedSplit*], None]* | *None* = *None*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*)

Bases: *StackedSplit*

A container which switches between children using expandable sections.

**property active: int | None**

Return the index of the active child container.

**active_child**() → AnyContainer

Return the currently active child container.

**add_style**(*style: str*) → str

Add a style to the widget's base style.

**property children: list[AnyContainer]**

Return a list of the widget's child containers.

**draw_container**() → None

Render the accordion in it's current state.

**load_container**() → AnyContainer

Create the accordiion widget's container.

**mouse_handler**(*index: int*, *mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

**refresh**() → None

Re-draw the container when the list of child containers changes.

**title_text**(*index: int*, *title: AnyFormattedText*) → StyleAndTextTuples

Generate the title for each child container.

**property titles: list[AnyFormattedText]**

Return the titles of the child containers.

**toggle**(*index: int*) → None

Toggle the visibility of a child container.

**class** euporie.core.widgets.layout.**Box**(*body: AnyContainer*, *padding: AnyDimension = None*, *padding_left: AnyDimension = None*, *padding_right: AnyDimension = None*, *padding_top: AnyDimension = None*, *padding_bottom: AnyDimension = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *char: None | str | Callable[[], str] = None*, *modal: bool = False*, *key_bindings:* KeyBindings | *None = None*)

Bases: object

Add padding around a container.

This also makes sure that the parent can provide more space than required by the child. This is very useful when wrapping a small element with a fixed size into a VSplit or HSplit object. The HSplit and VSplit try to make sure to adapt respectively the width and height, possibly shrinking other elements. Wrapping something in a Box makes it flexible.

**Parameters**

- **body** – Another container object.
- **padding** – The margin to be used around the body. This can be
- **padding_left** (*overridden by*) – *padding_bottom*.
- **padding_right`** – *padding_bottom*.
- **and** (*padding_top*) – *padding_bottom*.
- **style** – A style string.
- **char** – Character to be used for filling the space around the body. (This is supposed to be a character with a terminal width of 1.)

**class** euporie.core.widgets.layout.**ConditionalSplit**(*vertical: FilterOrBool*, **args: Any*, ***kwargs: Any*)

Bases: object

A split container where the orientation depends on a filter.

**container**() → _Split

Return the container for the current orientation.

**load_container**(*vertical: bool*) → _Split

Load the container.

**class** euporie.core.widgets.layout.**ReferencedSplit**(*split: type[_Split]*, *children: Sequence[AnyContainer]*, **args: Any*, ***kwargs: Any*)

Bases: object

A split container which maintains a reference to it's children.

**property children: list[AnyContainer]**

Convert the referenced children to containers.

**class** euporie.core.widgets.layout.**StackedSplit**(*children: Sequence[AnyContainer]*, *titles: Sequence[AnyFormattedText]*, *active: int = 0*, *style: str | Callable[[], str] = 'class:tab-split'*, *on_change: Callable[[StackedSplit], None] | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*)

Bases: object

Base class for containers with selectable children.

**property active: int | None**

Return the index of the active child container.

**active_child**() → AnyContainer

Return the currently active child container.

**add_style**(*style: str*) → str

Add a style to the widget's base style.

**property children: list[AnyContainer]**

Return a list of the widget's child containers.

**abstract load_container**() → AnyContainer

Abstract method for loading the widget's container.

**refresh**() → None

Reload the widget's container when its children or their titles change.

**property titles: list[AnyFormattedText]**

Return the titles of the child containers.

**class** euporie.core.widgets.layout.**TabBarControl**(*tabs: Sequence[TabBarTab] | Callable[[], Sequence[TabBarTab]]*, *active: int | Callable[[], int]*, *spacing: int = 1*, *closeable: bool = False*, *max_title_width: int = 30*)

Bases: *UIControl*

A control which shows a tab bar.

**property active: int**

Return the index of the active tab.

**char_bottom = '_'**

**char_close = '×'**

**char_left** = '| '

**char_right** = ' |'

**char_top** = '_'

**create_content**(*width: int*, *height: int*) → *UIContent*

Generate the formatted text fragments which make the controls output.

**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**() → *KeyBindingsBase* | None

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: GetLinePrefixCallable* | *None*) → int | None

Return the preferred height of the tab-bar control (2 rows).

**preferred_width**(*max_available_width: int*) → int | None

Return the preferred width of the tab-bar control, the maximum available.

**render**(*width: int*) → list[StyleAndTextTuples]

Render the tab-bar as linest of formatted text.

**reset**() → None

**property tabs: list[*euporie.core.widgets.layout.TabBarTab*]**

Return the tab-bar's tabs.

**class** euporie.core.widgets.layout.**TabBarTab**(*title: AnyFormattedText*, *on_activate: Callable*, *on_deactivate: Callable* | *None* = None, *on_close: Callable* | *None* = None)

Bases: NamedTuple

A named tuple represting a tab and it's callbacks.

**count**(*value*, */*)

Return number of occurrences of value.

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**on_activate: Callable**
Alias for field number 1

**on_close: Callable | None**
Alias for field number 3

**on_deactivate: Callable | None**
Alias for field number 2

**title: AnyFormattedText**
Alias for field number 0

**class** euporie.core.widgets.layout.**TabbedSplit**(*children: Sequence[AnyContainer], titles: Sequence[AnyFormattedText], active: int = 0, style: str | Callable[[], str] = 'class:tab-split', on_change: Callable[[StackedSplit], None] | None = None, width: AnyDimension = None, height: AnyDimension = None, border: GridStyle = □□□□ □ □□ □□□□ □□□□, show_borders: DiBool | None = None*)

Bases: *StackedSplit*

A container which switches between children using tabs.

**property active: int | None**
Return the index of the active child container.

**active_child**() → AnyContainer
Return the currently active child container.

**add_style**(*style: str*) → str
Add a style to the widget's base style.

**property children: list[AnyContainer]**
Return a list of the widget's child containers.

**load_container**() → AnyContainer
Create the tabbed widget's container.

Consists of a tab-bar control above a dynamic container which shows the active child container.

**Returns**
The widget's container

**load_tabs**() → list[*euporie.core.widgets.layout.TabBarTab*]
Return a list of tabs for the current children.

**refresh**() → None
Refresh the widget - set the tab-bar's tabs and active tab index.

**property titles: list[AnyFormattedText]**
Return the titles of the child containers.

### euporie.core.widgets.menu

Define an application menu.

### Functions

| | |
|---|---|
| *explode_text_fragments*(fragments) | Turn a list of (style_str, text) tuples into another list where each string is exactly one character. |
| *fragment_list_to_text*(fragments) | Concatenate all the text parts again. |
| *fragment_list_width*(fragments) | Return the character width of this text fragment list. |
| *get_app*() | Get the current active (running) Application. |
| *get_cwidth*(string) | Return width of a string. |
| *has_focus*(value) | Enable when this buffer has the focus. |
| *to_container*(container) | Make sure that the given object is a *Container*. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |
| *to_plain_text*(value) | Turn any kind of formatted text back into plain text. |

### euporie.core.widgets.menu.explode_text_fragments

euporie.core.widgets.menu.**explode_text_fragments**(*fragments: Iterable[_T]*) → _ExplodedList[_T]

Turn a list of (style_str, text) tuples into another list where each string is exactly one character.

It should be fine to call this function several times. Calling this on a list that is already exploded, is a null operation.

**Parameters**

**fragments** – List of (style, text) tuples.

### euporie.core.widgets.menu.fragment_list_to_text

euporie.core.widgets.menu.**fragment_list_to_text**(*fragments: StyleAndTextTuples*) → str

Concatenate all the text parts again.

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.menu.fragment_list_width

euporie.core.widgets.menu.**fragment_list_width**(*fragments: StyleAndTextTuples*) → int

Return the character width of this text fragment list. (Take double width characters into account.)

**Parameters**

**fragments** – List of `(style_str, text)` or `(style_str, text, mouse_handler)` tuples.

### euporie.core.widgets.menu.get_app

euporie.core.widgets.menu.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.menu.get_cwidth

euporie.core.widgets.menu.**get_cwidth**(*string: str*) → int

Return width of a string. Wrapper around `wcwidth`.

### euporie.core.widgets.menu.has_focus

euporie.core.widgets.menu.**has_focus**(*value: FocusableElement*) → *Condition*

Enable when this buffer has the focus.

### euporie.core.widgets.menu.to_container

euporie.core.widgets.menu.**to_container**(*container: AnyContainer*) → *Container*

Make sure that the given object is a *Container*.

### euporie.core.widgets.menu.to_filter

euporie.core.widgets.menu.**to_filter**(*bool_or_filter: Union[*Filter, *bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### euporie.core.widgets.menu.to_formatted_text

euporie.core.widgets.menu.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

### euporie.core.widgets.menu.to_plain_text

euporie.core.widgets.menu.**to_plain_text**(*value: AnyFormattedText*) → str

Turn any kind of formatted text back into plain text.

## Classes

| | |
|---|---|
| *CompletionsMenu*([max_height, scroll_offset, ...]) | A custom completions menu. |
| *CompletionsMenuControl*() | A custom completions menu control. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *Container*() | Base class for user interface layout. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *Float*(content[, top, right, bottom, left, ...]) | Float for use in a *FloatContainer*. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *KeyBindings*() | A container for a set of key bindings. |
| *MenuBar*(app, menu_items, grid) | A container to hold the menubar and main application body. |
| *MenuItem*([formatted_text, description, ...]) | A prompt-toolkit compatible menu item with more advanced capabilities. |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, sent to *UIControl.mouse_handler*. |
| *MouseEventType*(value[, names, module, ...]) | |
| *Point*(x, y) | |
| *PtkCompletionsMenuControl* | alias of CompletionsMenuControl |
| *ScrollOffsets*([top, bottom, left, right]) | Scroll offsets for the *Window* class. |
| *Shadow*(body) | Draw a shadow underneath/behind this container. |
| *StatusContainer*(body, status) | A container which allows attaching a status function. |
| *UIContent*(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.widgets.menu.CompletionsMenu

**class** euporie.core.widgets.menu.**CompletionsMenu**(*max_height: int | None = 16*, *scroll_offset: int | Callable[[], int] = 1*, *extra_filter: FilterOrBool = True*, *z_index: int = 100000000*)

A custom completions menu.

### euporie.core.widgets.menu.CompletionsMenuControl

**class** euporie.core.widgets.menu.**CompletionsMenuControl**

A custom completions menu control.

### euporie.core.widgets.menu.Condition

**class** euporie.core.widgets.menu.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.menu.ConditionalContainer

**class** euporie.core.widgets.menu.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.menu.Container

**class** euporie.core.widgets.menu.**Container**

Base class for user interface layout.

### euporie.core.widgets.menu.Dimension

**class** euporie.core.widgets.menu.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.

- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.widgets.menu.Float

**class** euporie.core.widgets.menu.**Float**(*content: AnyContainer*, *top: int | None = None*, *right: int | None = None*, *bottom: int | None = None*, *left: int | None = None*, *width: int | Callable[[], int] | None = None*, *height: int | Callable[[], int] | None = None*, *xcursor: bool = False*, *ycursor: bool = False*, *attach_to_window: AnyContainer | None = None*, *hide_when_covering_content: bool = False*, *allow_cover_cursor: bool = False*, *z_index: int = 1*, *transparent: bool = False*)

Float for use in a *FloatContainer*. Except for the *content* parameter, all other options are optional.

**Parameters**

- **content** – *Container* instance.
- **width** – *Dimension* or callable which returns a *Dimension*.
- **height** – *Dimension* or callable which returns a *Dimension*.
- **left** – Distance to the left edge of the *FloatContainer*.
- **right** – Distance to the right edge of the *FloatContainer*.
- **top** – Distance to the top of the *FloatContainer*.
- **bottom** – Distance to the bottom of the *FloatContainer*.
- **attach_to_window** – Attach to the cursor from this window, instead of the current window.
- **hide_when_covering_content** – Hide the float when it covers content underneath.
- **allow_cover_cursor** – When *False*, make sure to display the float below the cursor. Not on top of the indicated position.
- **z_index** – Z-index position. For a Float, this needs to be at least one. It is relative to the z_index of the parent container.
- **transparent** – *Filter* indicating whether this float needs to be drawn transparently.

### euporie.core.widgets.menu.FormattedTextControl

**class** euporie.core.widgets.menu.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **`focusable`** – *bool* or *`Filter`*: Tell whether this control is focusable.
- **`text`** – Text or formatted text to be displayed.
- **`style`** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **`key_bindings`** – a *`KeyBindings`* object.
- **`get_cursor_position`** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.widgets.menu.HSplit

**class** euporie.core.widgets.menu.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: VerticalAlign* = *VerticalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.menu.KeyBindings

**class** euporie.core.widgets.menu.**KeyBindings**

A container for a set of key bindings.

Example usage:

```
kb = KeyBindings()

@kb.add('c-t')
```

(continues on next page)

(continued from previous page)

```
def _(event):
    print('Control-T pressed')

@kb.add('c-a', 'c-b')
def _(event):
    print('Control-A pressed, followed by Control-B')

@kb.add('c-x', filter=is_searching)
def _(event):
    print('Control-X pressed')  # Works only if we are searching.
```

### euporie.core.widgets.menu.MenuBar

**class** euporie.core.widgets.menu.**MenuBar**(*app: BaseApp, menu_items: Sequence[MenuItem], grid: GridStyle = □□□□ □ □□ □□□□ □□□□*)

A container to hold the menubar and main application body.

### euporie.core.widgets.menu.MenuItem

**class** euporie.core.widgets.menu.**MenuItem**(*formatted_text: AnyFormattedText = ''*, *description: str = ''*, *separator: bool = False*, *handler: Callable[[], None] | None = None*, *children: list[*MenuItem*] | None = None*, *shortcut: AnyFormattedText = ''*, *hidden: FilterOrBool = False*, *disabled: FilterOrBool = False*, *toggled:* Filter *| None = None*, *collapse_prefix: bool = False*, *collapse_suffix: bool = True*)

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.

### euporie.core.widgets.menu.MouseEvent

**class** euporie.core.widgets.menu.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers: frozenset[*prompt_toolkit.mouse_events.MouseModifier*]*)

Mouse event, sent to *UIControl.mouse_handler*.

**Parameters**

- **position** – *Point* instance.
- **event_type** – *MouseEventType*.

### euporie.core.widgets.menu.MouseEventType

**class** euporie.core.widgets.menu.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.menu.Point

**class** euporie.core.widgets.menu.**Point**(*x*, *y*)

### euporie.core.widgets.menu.PtkCompletionsMenuControl

euporie.core.widgets.menu.**PtkCompletionsMenuControl**

alias of CompletionsMenuControl

### euporie.core.widgets.menu.ScrollOffsets

**class** euporie.core.widgets.menu.**ScrollOffsets**(*top: Union[int, Callable[[], int]] = 0*, *bottom: Union[int, Callable[[], int]] = 0*, *left: Union[int, Callable[[], int]] = 0*, *right: Union[int, Callable[[], int]] = 0*)

Scroll offsets for the *Window* class.

Note that left/right offsets only make sense if line wrapping is disabled.

### euporie.core.widgets.menu.Shadow

**class** euporie.core.widgets.menu.**Shadow**(*body: AnyContainer*)

Draw a shadow underneath/behind this container.

This is a globally configurable version of the prompt_toolkit.widows.base.Shadow class.

### euporie.core.widgets.menu.StatusContainer

**class** euporie.core.widgets.menu.**StatusContainer**(*body: AnyContainer*, *status: Callable[[], StatusBarFields | None]*)

A container which allows attaching a status function.

### euporie.core.widgets.menu.UIContent

**class** euporie.core.widgets.menu.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>, line_count: int = 0, cursor_position: Point | None = None, menu_position: Point | None = None, show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.widgets.menu.VSplit

**class** euporie.core.widgets.menu.**VSplit**(*children: Sequence[AnyContainer], window_too_small: Container | None = None, align: HorizontalAlign = HorizontalAlign.JUSTIFY, padding: AnyDimension = 0, padding_char: str | None = None, padding_style: str = '', width: AnyDimension = None, height: AnyDimension = None, z_index: int | None = None, modal: bool = False, key_bindings: KeyBindingsBase | None = None, style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.menu.Window

**class** euporie.core.widgets.menu.**Window**(*content: UIControl | None = None, width: AnyDimension = None, height: AnyDimension = None, z_index: int | None = None, dont_extend_width: FilterOrBool = False, dont_extend_height: FilterOrBool = False, ignore_content_width: FilterOrBool = False, ignore_content_height: FilterOrBool = False, left_margins: Sequence[Margin] | None = None, right_margins: Sequence[Margin] | None = None, scroll_offsets: ScrollOffsets | None = None, allow_scroll_beyond_bottom: FilterOrBool = False, wrap_lines: FilterOrBool = False, get_vertical_scroll: Callable[[Window], int] | None = None, get_horizontal_scroll: Callable[[Window], int] | None = None, always_hide_cursor: FilterOrBool = False, cursorline: FilterOrBool = False, cursorcolumn: FilterOrBool = False, colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None, align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT, style: str | Callable[[], str] = '', char: None | str | Callable[[], str] = None, get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.core.widgets.menu.partial

**class** euporie.core.widgets.menu.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.widgets.menu.**CompletionsMenu**(*max_height: int | None = 16*, *scroll_offset: int | Callable[[], int] = 1*, *extra_filter: FilterOrBool = True*, *z_index: int = 100000000*)

Bases: *ConditionalContainer*

A custom completions menu.

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

Return the list of child *Container* objects.

**get_key_bindings**() → *prompt_toolkit.key_binding.key_bindings.KeyBindingsBase* | None

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*) → *Dimension*

Return a Dimension that represents the desired height for this container.

**preferred_width**(*max_available_width: int*) → *Dimension*

Return a Dimension that represents the desired width for this container.

**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: int | None*) → None

Write the actual content to the screen.

**Parameters**

- **screen** – *Screen*
- **mouse_handlers** – *MouseHandlers*.
- **parent_style** – Style string to pass to the *Window* object. This will be applied to all content of the windows. *VSplit* and *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.core.widgets.menu.**CompletionsMenuControl**

Bases: CompletionsMenuControl

A custom completions menu control.

**MIN_WIDTH = 7**

**create_content**(*width: int*, *height: int*) → *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**() → *KeyBindingsBase* | None

The key bindings that are specific for this user control.

Return a `KeyBindings` object if some key bindings are specified, or *None* otherwise.

**has_focus**() → bool

**is_focusable**() → bool

Tell whether this user control is focusable.

**mouse_handler**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events: clicking and scrolling.

**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: GetLinePrefixCallable* | *None*) → int | None

**preferred_width**(*max_available_width: int*) → int | None

**reset**() → None

**class** euporie.core.widgets.menu.**MenuBar**(*app: BaseApp*, *menu_items: Sequence[MenuItem]*, *grid: GridStyle = ▯▯▯▯ ▯ ▯▯ ▯▯▯▯ ▯▯▯▯*)

Bases: `object`

A container to hold the menubar and main application body.

**refocus**() → None

Focus the currently selected menu.

**class** euporie.core.widgets.menu.**MenuItem**(*formatted_text: AnyFormattedText = ''*, *description: str = ''*, *separator: bool = False*, *handler: Callable[[], None]* | *None = None*, *children: list[*MenuItem*]* | *None = None*, *shortcut: AnyFormattedText = ''*, *hidden: FilterOrBool = False*, *disabled: FilterOrBool = False*, *toggled:* Filter | *None = None*, *collapse_prefix: bool = False*, *collapse_suffix: bool = True*)

Bases: `object`

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**

Return 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 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.

## euporie.core.widgets.pager

Contain a container for the cell output area.

### Functions

| | |
|---|---|
| *NamedTuple*(typename[, fields]) | Typed version of namedtuple. |
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *find_route*(from_, to) | Find the shortest conversion path between two formats. |
| *get_app*() | Get the current active (running) Application. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *register_bindings*(bindings) | Update the key-binding registry. |

### euporie.core.widgets.pager.NamedTuple

euporie.core.widgets.pager.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)

Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

### euporie.core.widgets.pager.add_cmd

euporie.core.widgets.pager.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.widgets.pager.find_route

euporie.core.widgets.pager.**find_route**(*from_: str*, *to: str*) → list | None

Find the shortest conversion path between two formats.

### euporie.core.widgets.pager.get_app

euporie.core.widgets.pager.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.pager.load_registered_bindings

euporie.core.widgets.pager.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.core.widgets.pager.register_bindings

euporie.core.widgets.pager.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

## Classes

| | |
|---|---|
| *Box*(body[, padding, padding_left, ...]) | Add padding around a container. |
| *CellOutput*(json, parent) | Represent a single cell output. |
| *CellOutputDataElement*(mime, data, metadata, ...) | A cell output element which display data. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *Datum*(data, *args, **kwargs) | Class for storing and converting display data. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *Display*(datum[, height, width, focusable, ...]) | Rich output displays. |
| *DynamicContainer*(get_container) | Container class that dynamically returns any Container. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *Line*([char, width, height, collapse, style]) | Draw a horizontal or vertical line. |
| *Pager*([height]) | Interactive help pager. |
| *PagerOutput*(json, parent) | Display pager output. |
| *PagerOutputDataElement*(mime, data, metadata, ...) | A cell output element which display data. |
| *PagerState*(code, cursor_pos, data) | A named tuple which describes the state of a pager. |
| *PurePath*(*args, **kwargs) | Base class for manipulating paths without I/O. |

### euporie.core.widgets.pager.Box

**class** euporie.core.widgets.pager.**Box**(*body: AnyContainer*, *padding: AnyDimension = None*, *padding_left: AnyDimension = None*, *padding_right: AnyDimension = None*, *padding_top: AnyDimension = None*, *padding_bottom: AnyDimension = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *char: None | str | Callable[[], str] = None*, *modal: bool = False*, *key_bindings:* KeyBindings | *None = None*)

Add padding around a container.

This also makes sure that the parent can provide more space than required by the child. This is very useful when wrapping a small element with a fixed size into a `VSplit` or `HSplit` object. The `HSplit` and `VSplit` try to make sure to adapt respectively the width and height, possibly shrinking other elements. Wrapping something in a `Box` makes it flexible.

**Parameters**

- **body** – Another container object.
- **padding** – The margin to be used around the body. This can be
- **padding_left** (*overridden by*) – *padding_bottom*.
- **padding_right`** – *padding_bottom*.

- **and** (*padding_top*) – *padding_bottom*.
- **style** – A style string.
- **char** – Character to be used for filling the space around the body. (This is supposed to be a character with a terminal width of 1.)

### euporie.core.widgets.pager.CellOutput

**class** euporie.core.widgets.pager.**CellOutput**(*json: dict[str, Any]*, *parent: OutputParent | None*)

Represent a single cell output.

Capable of displaying multiple mime representations of the same data.

TODO - allow the visible mime-type to be rotated.

### euporie.core.widgets.pager.CellOutputDataElement

**class** euporie.core.widgets.pager.**CellOutputDataElement**(*mime: str*, *data: Any*, *metadata: dict*, *parent: OutputParent | None*)

A cell output element which display data.

### euporie.core.widgets.pager.Condition

**class** euporie.core.widgets.pager.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.pager.ConditionalContainer

**class** euporie.core.widgets.pager.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.pager.Datum

**class** euporie.core.widgets.pager.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Class for storing and converting display data.

### euporie.core.widgets.pager.Dimension

**class** euporie.core.widgets.pager.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.core.widgets.pager.Display

**class** euporie.core.widgets.pager.**Display**(*datum: Datum*, *height: AnyDimension = None*, *width: AnyDimension = None*, *focusable: FilterOrBool = False*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *always_hide_cursor: FilterOrBool = True*, *scrollbar: FilterOrBool = True*, *scrollbar_autohide: FilterOrBool = True*, *dont_extend_height: FilterOrBool = True*, *dont_extend_width: FilterOrBool = False*, *style: str | Callable[[], str] = ''*)

Rich output displays.

A container for displaying rich output data.

### euporie.core.widgets.pager.DynamicContainer

**class** euporie.core.widgets.pager.**DynamicContainer**(*get_container: Callable[[], AnyContainer]*)

Container class that dynamically returns any Container.

**Parameters**

**get_container** – Callable that returns a *Container* instance or any widget with a `__pt_container__` method.

### euporie.core.widgets.pager.HSplit

**class** euporie.core.widgets.pager.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.pager.Line

**class** euporie.core.widgets.pager.**Line**(*char: str | None = None*, *width: int | None = None*, *height: int | None = None*, *collapse: bool = False*, *style: str = 'class:grid-line'*)

Draw a horizontal or vertical line.

### euporie.core.widgets.pager.Pager

**class** euporie.core.widgets.pager.**Pager**(*height: AnyDimension | None = None*)

Interactive help pager.

A pager which displays information at the bottom of a tab.

### euporie.core.widgets.pager.PagerOutput

**class** euporie.core.widgets.pager.**PagerOutput**(*json: dict[str, Any]*, *parent: OutputParent | None*)

Display pager output.

### euporie.core.widgets.pager.PagerOutputDataElement

**class** euporie.core.widgets.pager.**PagerOutputDataElement**(*mime: str*, *data: dict[str, Any]*, *metadata: dict*, *parent: OutputParent | None*)

A cell output element which display data.

### euporie.core.widgets.pager.PagerState

**class** euporie.core.widgets.pager.**PagerState**(*code: str*, *cursor_pos: int*, *data: dict*)

A named tuple which describes the state of a pager.

### euporie.core.widgets.pager.PurePath

**class** euporie.core.widgets.pager.**PurePath**(**args*, ***kwargs*)

Base class for manipulating paths without I/O.

PurePath represents a filesystem path and offers operations which don't imply any actual filesystem I/O. Depending on your system, instantiating a PurePath will return either a PurePosixPath or a PureWindowsPath object. You can also instantiate either of these classes directly, regardless of your system.

**class** euporie.core.widgets.pager.**Pager**(*height: AnyDimension | None = None*)

Bases: object

Interactive help pager.

A pager which displays information at the bottom of a tab.

**focus**() → None

Focus the pager.

**hide**() → None

Clear and hide the pager.

**property state: *euporie.core.widgets.pager.PagerState* | None**

Return the pager's current state.

**class** euporie.core.widgets.pager.**PagerOutput**(*json: dict[str, Any]*, *parent: OutputParent | None*)

Bases: *CellOutput*

Display pager output.

**property data: dict[str, Any]**

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.

**property element: *CellOutputElement***

Get the element for the currently selected mime type.

**get_element**(*mime: str*) → *CellOutputElement*

Return the currently displayed cell element.

**make_element**(*mime: str*) → *CellOutputElement*

Create a container for the pager output mime-type if it doesn't exist.

**Parameters**

**mime** – The mime-type to make the container for

**Returns**

A *PagerOutputDataElement* container for the currently selected mime-type.

**scroll_left**() → None

Scroll the currently visible output left.

**scroll_right**() → None

Scroll the currently visible output right.

**property selected_mime: str**

Return the selected mime-type, selecting the first by default.

**update**() → None

Update the output by updating all child containers.

**class** euporie.core.widgets.pager.**PagerOutputDataElement**(*mime: str*, *data: dict[str, Any]*, *metadata: dict*, *parent: OutputParent | None*)

Bases: *CellOutputDataElement*

A cell output element which display data.

**property data: Any**

Return the control's display data.

**scroll_left**() → None

Scroll the output left.

**scroll_right**() → None

Scroll the output right.

**class** euporie.core.widgets.pager.**PagerState**(*code: str*, *cursor_pos: int*, *data: dict*)

Bases: NamedTuple

A named tuple which describes the state of a pager.

**code: str**

Alias for field number 0

**count**(*value*, */*)

Return number of occurrences of value.

**cursor_pos: int**

Alias for field number 1

**data: dict**

Alias for field number 2

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

### euporie.core.widgets.palette

Contain the command palette container.

### Functions

| | |
|---|---|
| *NamedTuple*(typename[, fields]) | Typed version of namedtuple. |
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *focus_next*(event) | Focus the next visible Window. |
| *focus_previous*(event) | Focus the previous visible Window. |
| *get_app*() | Get the current active (running) Application. |
| *register_bindings*(bindings) | Update the key-binding registry. |

### euporie.core.widgets.palette.NamedTuple

euporie.core.widgets.palette.**NamedTuple**(*typename*, *fields=None*, */*, ***kwargs*)

Typed version of namedtuple.

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has an extra __annotations__ attribute, giving a dict that maps field names to types. (The field names are also in the _fields attribute, which is part of the namedtuple API.) An alternative equivalent functional syntax is also accepted:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

### euporie.core.widgets.palette.add_cmd

euporie.core.widgets.palette.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.widgets.palette.focus_next

euporie.core.widgets.palette.**focus_next**(*event:* KeyPressEvent) → None

Focus the next visible Window. (Often bound to the *Tab* key.)

### euporie.core.widgets.palette.focus_previous

euporie.core.widgets.palette.**focus_previous**(*event:* KeyPressEvent) → None

Focus the previous visible Window. (Often bound to the *BackTab* key.)

### euporie.core.widgets.palette.get_app

euporie.core.widgets.palette.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.palette.register_bindings

euporie.core.widgets.palette.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### Classes

| | |
|---|---|
| *Command*(handler, *, filter, hidden, name, ...) | Wrap a function so it can be used as a key-binding or a menu item. |
| *CommandMenuControl*(command_palette[, width]) | A filterable and navigable list of commands. |
| *CommandPalette*(app) | A command palette which allows searching the available commands. |
| *Condition*(func) | Turn any callable into a Filter. |
| *Dialog*(app) | A modal dialog which is displayed above the application. |
| *FocusedStyle*(body[, style_focus, style_hover]) | Apply a style to child containers when focused or hovered. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *MarginContainer*(margin, target) | A container which renders a stand-alone margin. |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, sent to *UIControl.mouse_handler*. |
| *MouseEventType*(value[, names, module, ...]) | |
| *Point*(x, y) | |
| *ScrollOffsets*([top, bottom, left, right]) | Scroll offsets for the *Window* class. |
| *ScrollbarMargin*([display_arrows, ...]) | Margin displaying a scrollbar. |
| *StatusContainer*(body, status) | A container which allows attaching a status function. |
| *Text*([text, style, height, min_height, ...]) | A text input widget. |
| *UIContent*(get_line, StyleAndTextTuples] = >, ...) | Content generated by a user control. |
| *UIControl*() | Base class for all user interface controls. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.core.widgets.palette.Command

**class** euporie.core.widgets.palette.**Command**(*handler: CommandHandler, *, filter: FilterOrBool = True, hidden: FilterOrBool = False, name: str | None = None, title: str | None = None, menu_title: str | None = None, description: str | None = None, toggled: Filter | None = None, eager: FilterOrBool = False, is_global: FilterOrBool = False, save_before: Callable[[KeyPressEvent], bool] = <function Command.<lambda>>, record_in_macro: FilterOrBool = True*)

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

### euporie.core.widgets.palette.CommandMenuControl

**class** euporie.core.widgets.palette.**CommandMenuControl**(*command_palette:* CommandPalette, *width: int = 60*)

A filterable and navigable list of commands.

### euporie.core.widgets.palette.CommandPalette

**class** euporie.core.widgets.palette.**CommandPalette**(*app:* BaseApp)

A command palette which allows searching the available commands.

### euporie.core.widgets.palette.Condition

**class** euporie.core.widgets.palette.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.palette.Dialog

**class** euporie.core.widgets.palette.**Dialog**(*app:* BaseApp)

A modal dialog which is displayed above the application.

Returns focus to the previously selected control when closed.

### euporie.core.widgets.palette.FocusedStyle

**class** euporie.core.widgets.palette.**FocusedStyle**(*body: AnyContainer*, *style_focus: str | Callable[[], str] = 'class:focused'*, *style_hover: str | Callable[[], str] = ''*)

Apply a style to child containers when focused or hovered.

### euporie.core.widgets.palette.HSplit

**class** euporie.core.widgets.palette.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.palette.MarginContainer

**class** euporie.core.widgets.palette.**MarginContainer**(*margin: Margin*, *target: ScrollableContainer*)

A container which renders a stand-alone margin.

### euporie.core.widgets.palette.MouseEvent

**class** euporie.core.widgets.palette.**MouseEvent**(*position: Point*, *event_type: MouseEventType*, *button: MouseButton*, *modifiers: frozenset[*prompt_toolkit.mouse_events.MouseModifier*]*)

Mouse event, sent to *UIControl.mouse_handler*.

**Parameters**

- **position** – *Point* instance.
- **event_type** – *MouseEventType*.

### euporie.core.widgets.palette.MouseEventType

**class** euporie.core.widgets.palette.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.palette.Point

**class** euporie.core.widgets.palette.**Point**(*x*, *y*)

### euporie.core.widgets.palette.ScrollOffsets

**class** euporie.core.widgets.palette.**ScrollOffsets**(*top: Union[int, Callable[[], int]] = 0*, *bottom: Union[int, Callable[[], int]] = 0*, *left: Union[int, Callable[[], int]] = 0*, *right: Union[int, Callable[[], int]] = 0*)

Scroll offsets for the *Window* class.

Note that left/right offsets only make sense if line wrapping is disabled.

### euporie.core.widgets.palette.ScrollbarMargin

**class** euporie.core.widgets.palette.**ScrollbarMargin**(*display_arrows: Union[Filter, bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[Filter, bool] = False*, *smooth: bool = True*, *style: str = ''*)

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

### euporie.core.widgets.palette.StatusContainer

**class** euporie.core.widgets.palette.**StatusContainer**(*body: AnyContainer*, *status: Callable[[], StatusBarFields | None]*)

A container which allows attaching a status function.

### euporie.core.widgets.palette.Text

**class** euporie.core.widgets.palette.**Text**(*text: str = ''*, *style: str = 'class:input'*, *height: int = 1*, *min_height: int = 1*, *multiline: FilterOrBool = False*, *expand: FilterOrBool = True*, *width: int | None = None*, *completer: Completer | None = None*, *options: list[str] | Callable[[], list[str]] | None = None*, *show_borders: DiBool | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *validation: Callable[[str], bool] | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *placeholder: str | None = None*, *lexer: Lexer | None = None*, *input_processors: Sequence[Processor] | None = None*, *disabled: FilterOrBool = False*, *password: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *prompt: AnyFormattedText | None = None*)

A text input widget.

### euporie.core.widgets.palette.UIContent

**class** euporie.core.widgets.palette.**UIContent**(*get_line: Callable[[int], StyleAndTextTuples] = <function UIContent.<lambda>>*, *line_count: int = 0*, *cursor_position: Point | None = None*, *menu_position: Point | None = None*, *show_cursor: bool = True*)

Content generated by a user control. This content consists of a list of lines.

**Parameters**

- **get_line** – Callable that takes a line number and returns the current line. This is a list of (style_str, text) tuples.
- **line_count** – The number of lines.
- **cursor_position** – a *Point* for the cursor position.
- **menu_position** – a *Point* for the menu position.
- **show_cursor** – Make the cursor visible.

### euporie.core.widgets.palette.UIControl

**class** euporie.core.widgets.palette.**UIControl**

Base class for all user interface controls.

### euporie.core.widgets.palette.VSplit

**class** euporie.core.widgets.palette.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.palette.Window

**class** euporie.core.widgets.palette.**Window**(*content:* UIControl | *None* = *None*, *width:* *AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *dont_extend_width:* *FilterOrBool* = *False*, *dont_extend_height: FilterOrBool* = *False*, *ignore_content_width: FilterOrBool* = *False*, *ignore_content_height: FilterOrBool* = *False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool* = *False*, *wrap_lines: FilterOrBool* = *False*, *get_vertical_scroll:* *Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool* = *False*, *cursorline: FilterOrBool* = *False*, *cursorcolumn:* *FilterOrBool* = *False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]]* = *None*, *align:* WindowAlign | *Callable[[],* WindowAlign*]* = *WindowAlign.LEFT*, *style: str* | *Callable[[], str]* = '', *char: None* | *str* | *Callable[[], str]* = *None*, *get_line_prefix: GetLinePrefixCallable* | *None* = *None*)

Container that holds a control.

### euporie.core.widgets.palette.partial

**class** euporie.core.widgets.palette.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.core.widgets.palette.**CommandMenuControl**(*command_palette:* CommandPalette, *width: int* = *60*)

Bases: *UIControl*

A filterable and navigable list of commands.

**create_content**(*width: int*, *height: int*) → *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**() → *KeyBindingsBase* | None

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: GetLinePrefixCallable | None*) → int | None

Return the preferred height of the command list.

**preferred_width**(*max_available_width: int*) → int | None

Return the preferred width of the command list.

**reset**() → None

**class** euporie.core.widgets.palette.**CommandPalette**(*app:* BaseApp)

Bases: *Dialog*

A command palette which allows searching the available commands.

**accept**(*buffer:* Buffer | *None* = *None*) → bool

Call on `enter`: runs the selected command.

**body: AnyContainer**

**body_padding_bottom = 0**

**body_padding_top = 0**

**button_widgets: list[AnyContainer]**

**buttons: dict[str, Callable | None]**

**get_height**() → int | None

**get_width**() → int | None

**hide**(*event:* KeyPressEvent | *None* = *None*) → None

Hide the dialog.

**index: int**

**last_focused: *UIControl* | None**

**load**() → None

Reset the dialog ready for display.

**matches: list[_CommandMatch]**

**select**(*n: int*, *event:* KeyPressEvent | *None* = *None*) → None

Change the index of the selected command.

**Parameters**

- **n** – The relative amount by which to change the selected index

- **event** – Ignored

**show**(***params: Any*) → None

Display and focuses the dialog.

**text_changed**(*buffer:* Buffer) → None

Call when the input text changes: filters the command list.

**title: AnyFormattedText | None = 'Search for a command'**

**to_focus: FocusableElement | None**

**toggle**() → None

Show or hides the dialog.

### euporie.core.widgets.search

Define the global search toolbar and related search functions.

### Functions

| | |
|---|---|
| *accept_search*() | Accept the search input. |
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *find*() | Enter search mode. |
| *find_next*() | Find the next search match. |
| *find_prev_next*(direction) | Find the previous or next search match. |
| *find_previous*() | Find the previous search match. |
| *find_search_control*() | Find the current search buffer and buffer control. |
| *find_searchable_controls*(...) | Find list of searchable controls and the index of the next control. |
| *get_app*() | Get the current active (running) Application. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *start_global_search*([buffer_control, direction]) | Start a search through all searchable *buffer_controls* in the layout. |
| *stop_search*() | Abort the search. |

### euporie.core.widgets.search.accept_search

euporie.core.widgets.search.**accept_search**() → None

Accept the search input.

### euporie.core.widgets.search.add_cmd

euporie.core.widgets.search.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.core.widgets.search.find

euporie.core.widgets.search.**find**() → None

Enter search mode.

### euporie.core.widgets.search.find_next

euporie.core.widgets.search.**find_next**() → None

Find the next search match.

### euporie.core.widgets.search.find_prev_next

euporie.core.widgets.search.**find_prev_next**(*direction:* SearchDirection) → None

Find the previous or next search match.

### euporie.core.widgets.search.find_previous

euporie.core.widgets.search.**find_previous**() → None

Find the previous search match.

### euporie.core.widgets.search.find_search_control

euporie.core.widgets.search.**find_search_control**() → tuple[*prompt_toolkit.layout.controls.SearchBufferControl* | None, *prompt_toolkit.layout.controls.BufferControl* | None]

Find the current search buffer and buffer control.

### euporie.core.widgets.search.find_searchable_controls

euporie.core.widgets.search.**find_searchable_controls**(*search_buffer_control:* SearchBufferControl, *current_control:* prompt_toolkit.layout.controls.BufferControl | *None*) → list[*prompt_toolkit.layout.controls.BufferControl*]

Find list of searchable controls and the index of the next control.

### euporie.core.widgets.search.get_app

euporie.core.widgets.search.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.search.load_registered_bindings

euporie.core.widgets.search.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.core.widgets.search.register_bindings

euporie.core.widgets.search.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.core.widgets.search.start_global_search

euporie.core.widgets.search.**start_global_search**(*buffer_control:* prompt_toolkit.layout.controls.BufferControl | *None* = *None*, *direction:* SearchDirection = *SearchDirection.FORWARD*) → None

Start a search through all searchable *buffer_controls* in the layout.

### euporie.core.widgets.search.stop_search

euporie.core.widgets.search.**stop_search**() → None

Abort the search.

## Classes

| | |
|---|---|
| *BufferControl*([buffer, input_processors, ...]) | Control for visualizing the content of a *Buffer*. |
| *Condition*(func) | Turn any callable into a Filter. |
| *Document*([text, cursor_position, selection]) | This is a immutable class around the text and cursor position, and contains methods for querying this data, e.g. |
| *InputMode*(value[, names, module, qualname, ...]) | |
| *PtkSearchToolbar* | alias of *SearchToolbar* |
| *SearchBar*([search_buffer, vi_mode, ...]) | Search mode. |
| *SearchBufferControl*([buffer, ...]) | *BufferControl* which is used for searching another *BufferControl*. |
| *SearchDirection*(value[, names, module, ...]) | |
| *SelectionState*([original_cursor_position, type]) | State of the current selection. |

### euporie.core.widgets.search.BufferControl

**class** euporie.core.widgets.search.**BufferControl**(*buffer:* Buffer | *None* = *None*, *input_processors: list[*Processor*]* | *None* = *None*, *include_default_input_processors:* *bool* = *True*, *lexer:* *Lexer* | *None* = *None*, *preview_search:* *FilterOrBool* = *False*, *focusable:* *FilterOrBool* = *True*, *search_buffer_control:* *None* | SearchBufferControl | *Callable[[],* SearchBufferControl*]* = *None*, *menu_position: Callable[[],* *int* | *None*] | *None* = *None*, *focus_on_click:* *FilterOrBool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*)

Control for visualizing the content of a *Buffer*.

**Parameters**

- **buffer** – The *Buffer* object to be displayed.
- **input_processors** – A list of *Processor* objects.
- **include_default_input_processors** – When True, include the default processors for highlighting of selection, search and displaying of multiple cursors.
- **lexer** – Lexer instance for syntax highlighting.
- **preview_search** – *bool* or *Filter*: Show search while typing. When this is *True*, probably you want to add a HighlightIncrementalSearchProcessor as well. Otherwise only the cursor position will move, but the text won't be highlighted.
- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **focus_on_click** – Focus this buffer when it's click, but not yet focused.
- **key_bindings** – a *KeyBindings* object.

### euporie.core.widgets.search.Condition

**class** euporie.core.widgets.search.**Condition**(*func: Callable[[],* *bool*])

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.search.Document

**class** euporie.core.widgets.search.**Document**(*text: str = ''*, *cursor_position: int | None = None*, *selection:* prompt_toolkit.selection.SelectionState | *None = None*)

This is a immutable class around the text and cursor position, and contains methods for querying this data, e.g. to give the text before the cursor.

This class is usually instantiated by a *Buffer* object, and accessed as the *document* property of that class.

**Parameters**

- **text** – string
- **cursor_position** – int
- **selection** – *SelectionState*

### euporie.core.widgets.search.InputMode

**class** euporie.core.widgets.search.**InputMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.search.PtkSearchToolbar

euporie.core.widgets.search.**PtkSearchToolbar**

alias of *SearchToolbar*

### euporie.core.widgets.search.SearchBar

**class** euporie.core.widgets.search.**SearchBar**(*search_buffer:* Buffer | *None = None*, *vi_mode: bool = False*, *text_if_not_searching: AnyFormattedText = ''*, *forward_search_prompt: AnyFormattedText = 'I-search: '*, *backward_search_prompt: AnyFormattedText = 'I-search backward: '*, *ignore_case: FilterOrBool = False*)

Search mode.

A search toolbar with custom style and text.

### euporie.core.widgets.search.SearchBufferControl

**class** euporie.core.widgets.search.**SearchBufferControl**(*buffer:* Buffer | *None = None*, *input_processors: list[*Processor*]* | *None = None*, *lexer: Lexer | None = None*, *focus_on_click: FilterOrBool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *ignore_case: FilterOrBool = False*)

*BufferControl* which is used for searching another *BufferControl*.

**Parameters**
**ignore_case** – Search case insensitive.

### euporie.core.widgets.search.SearchDirection

**class** euporie.core.widgets.search.**SearchDirection**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.search.SelectionState

**class** euporie.core.widgets.search.**SelectionState**(*original_cursor_position: int = 0*, *type: SelectionType = SelectionType.CHARACTERS*)

State of the current selection.

**Parameters**

- **original_cursor_position** – int
- **type** – *SelectionType*

**class** euporie.core.widgets.search.**SearchBar**(*search_buffer: Buffer | None = None*, *vi_mode: bool = False*, *text_if_not_searching: AnyFormattedText = ''*, *forward_search_prompt: AnyFormattedText = 'I-search: '*, *backward_search_prompt: AnyFormattedText = 'I-search backward: '*, *ignore_case: FilterOrBool = False*)

Bases: *SearchToolbar*

Search mode.

A search toolbar with custom style and text.

euporie.core.widgets.search.**accept_search**() → None

Accept the search input.

euporie.core.widgets.search.**find**() → None

Enter search mode.

euporie.core.widgets.search.**find_next**() → None

Find the next search match.

euporie.core.widgets.search.**find_prev_next**(*direction:* SearchDirection) → None

Find the previous or next search match.

euporie.core.widgets.search.**find_previous**() → None

Find the previous search match.

euporie.core.widgets.search.**find_search_control**() → tuple[*prompt_toolkit.layout.controls.SearchBufferControl* | None, *prompt_toolkit.layout.controls.BufferControl* | None]

Find the current search buffer and buffer control.

euporie.core.widgets.search.**find_searchable_controls**(*search_buffer_control:* SearchBufferControl, *current_control:* prompt_toolkit.layout.controls.BufferControl | *None*) → list[*prompt_toolkit.layout.controls.BufferControl*]

Find list of searchable controls and the index of the next control.

euporie.core.widgets.search.**start_global_search**(*buffer_control:* prompt_toolkit.layout.controls.BufferControl | *None* = *None*, *direction:* SearchDirection = *SearchDirection.FORWARD*) → None

Start a search through all searchable *buffer_controls* in the layout.

euporie.core.widgets.search.**stop_search**() → None

Abort the search.

### euporie.core.widgets.status

Define a status-bar widget.

### Functions

| | |
|---|---|
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *get_app*() | Get the current active (running) Application. |
| *ptk_to_container*(container) | Make sure that the given object is a *Container*. |
| *to_container*(container) | Monkey-patch *to_container* to collect container status functions. |
| *to_filter*(bool_or_filter) | Accept both booleans and Filters as input and turn it into a Filter. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |

### euporie.core.widgets.status.add_setting

euporie.core.widgets.status.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.core.widgets.status.get_app

euporie.core.widgets.status.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.core.widgets.status.ptk_to_container

euporie.core.widgets.status.**ptk_to_container**(*container: AnyContainer*) → *Container*

Make sure that the given object is a *Container*.

### euporie.core.widgets.status.to_container

euporie.core.widgets.status.**to_container**(*container: AnyContainer*) → *Container*

Monkey-patch *to_container* to collect container status functions.

### euporie.core.widgets.status.to_filter

euporie.core.widgets.status.**to_filter**(*bool_or_filter: Union[*Filter*, bool]*) → *Filter*

Accept both booleans and Filters as input and turn it into a Filter.

### euporie.core.widgets.status.to_formatted_text

euporie.core.widgets.status.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

## Classes

| | |
|---|---|
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *FastDictCache*(get_value[, size]) | Fast, lightweight cache which keeps at most *size* items. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *StatusBar*([extra_filter, default]) | A status bar which shows the status of the current tab. |
| *StatusContainer*(body, status) | A container which allows attaching a status function. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *WeakKeyDictionary*([dict]) | Mapping class that references keys weakly. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *WindowAlign*(value[, names, module, ...]) | Alignment of the Window content. |

### euporie.core.widgets.status.ConditionalContainer

**class** euporie.core.widgets.status.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.status.FastDictCache

**class** euporie.core.widgets.status.**FastDictCache**(*get_value: Callable[[...], _V]*, *size: int = 1000000*)

Fast, lightweight cache which keeps at most *size* items. It will discard the oldest items in the cache first.

The cache is a dictionary, which doesn't keep track of access counts. It is perfect to cache little immutable objects which are not expensive to create, but where a dictionary lookup is still much faster than an object instantiation.

**Parameters**

**get_value** – Callable that's called in case of a missing key.

### euporie.core.widgets.status.FormattedTextControl

**class** euporie.core.widgets.status.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

> The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole Window, pass the style to the Window instead.)
- **key_bindings** – a *KeyBindings* object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.widgets.status.StatusBar

**class** euporie.core.widgets.status.**StatusBar**(*extra_filter: FilterOrBool = True*, *default: StatusBarFields | None = None*)

A status bar which shows the status of the current tab.

### euporie.core.widgets.status.StatusContainer

**class** euporie.core.widgets.status.**StatusContainer**(*body: AnyContainer*, *status: Callable[[], StatusBarFields | None]*)

A container which allows attaching a status function.

### euporie.core.widgets.status.VSplit

**class** euporie.core.widgets.status.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.status.WeakKeyDictionary

**class** euporie.core.widgets.status.**WeakKeyDictionary**(*dict=None*)

Mapping class that references keys weakly.

Entries in the dictionary will be discarded when there is no longer a strong reference to the key. This can be used to associate additional data with an object owned by other parts of an application without adding attributes to those objects. This can be especially useful with objects that override attribute accesses.

### euporie.core.widgets.status.Window

**class** euporie.core.widgets.status.**Window**(*content:* UIControl | *None* = *None*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *dont_extend_width: FilterOrBool* = *False*, *dont_extend_height: FilterOrBool* = *False*, *ignore_content_width: FilterOrBool* = *False*, *ignore_content_height: FilterOrBool* = *False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool* = *False*, *wrap_lines: FilterOrBool* = *False*, *get_vertical_scroll: Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool* = *False*, *cursorline: FilterOrBool* = *False*, *cursorcolumn: FilterOrBool* = *False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]]* = *None*, *align:* WindowAlign | *Callable[[],* WindowAlign*]* = *WindowAlign.LEFT*, *style: str* | *Callable[[], str]* = '', *char: None* | *str* | *Callable[[], str]* = *None*, *get_line_prefix: GetLinePrefixCallable* | *None* = *None*)

Container that holds a control.

### euporie.core.widgets.status.WindowAlign

**class** euporie.core.widgets.status.**WindowAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of the Window content.

Note that this is different from *HorizontalAlign* and *VerticalAlign*, which are used for the alignment of the child containers in respectively *VSplit* and *HSplit*.

**class** euporie.core.widgets.status.**StatusBar**(*extra_filter: FilterOrBool* = *True*, *default: StatusBarFields* | *None* = *None*)

Bases: object

A status bar which shows the status of the current tab.

**class** euporie.core.widgets.status.**StatusContainer**(*body: AnyContainer*, *status: Callable[[], StatusBarFields* | *None]*)

Bases: object

A container which allows attaching a status function.

euporie.core.widgets.status.**to_container**(*container: AnyContainer*) → *Container*

Monkey-patch *to_container* to collect container status functions.

### euporie.core.widgets.tree

Define a tree-view widget.

### Functions

| | |
|---|---|
| *cast*(typ, val) | Cast a value to a type. |

### euporie.core.widgets.tree.cast

euporie.core.widgets.tree.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### Classes

| | |
|---|---|
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *JsonView*(data[, title, expanded]) | A JSON-view container. |
| *MouseButton*(value[, names, module, ...]) | |
| *MouseEvent*(position, event_type, button, ...) | Mouse event, sent to *UIControl.mouse_handler*. |
| *MouseEventType*(value[, names, module, ...]) | |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |

### euporie.core.widgets.tree.Condition

**class** euporie.core.widgets.tree.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.core.widgets.tree.ConditionalContainer

**class** euporie.core.widgets.tree.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.core.widgets.tree.FormattedTextControl

**class** euporie.core.widgets.tree.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[],* Point | *None]* | *None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **key_bindings** – a *KeyBindings* object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.core.widgets.tree.HSplit

**class** euporie.core.widgets.tree.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: VerticalAlign* = *VerticalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked above/under the other.

### euporie.core.widgets.tree.JsonView

**class** euporie.core.widgets.tree.**JsonView**(*data: Any*, *title: str* | *None* = *None*, *expanded: bool* = *True*)

A JSON-view container.

### euporie.core.widgets.tree.MouseButton

**class** euporie.core.widgets.tree.**MouseButton**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.tree.MouseEvent

**class** euporie.core.widgets.tree.**MouseEvent**(*position:* Point, *event_type:* MouseEventType, *button:* MouseButton, *modifiers: frozenset[*prompt_toolkit.mouse_events.MouseModifier*]*)

Mouse event, sent to *UIControl.mouse_handler*.

**Parameters**

- **position** – *Point* instance.
- **event_type** – *MouseEventType*.

### euporie.core.widgets.tree.MouseEventType

**class** euporie.core.widgets.tree.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.core.widgets.tree.VSplit

**class** euporie.core.widgets.tree.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.core.widgets.tree.Window

**class** euporie.core.widgets.tree.**Window**(*content:* UIControl | *None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[*Margin*] | None = None*, *right_margins: Sequence[*Margin*] | None = None*, *scroll_offsets:* ScrollOffsets | *None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[*Window*], int] | None = None*, *get_horizontal_scroll: Callable[[*Window*], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align:* WindowAlign | *Callable[[],* WindowAlign*] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

**class** euporie.core.widgets.tree.**JsonView**(*data: Any*, *title: str | None = None*, *expanded: bool = True*)

Bases: object

A JSON-view container.

**format_title**() → StyleAndTextTuples

Return the tree node toggle and title.

**toggle**(*mouse_event:* MouseEvent) → NotImplementedOrNone

Toggle the expansion state.

## 5.12.3 euporie.hub

A multi-user hub euporie application.

### Modules

| | |
|---|---|
| *euporie.hub.app* | Run euporie as a multi-client SSH server. |

### euporie.hub.app

Run euporie as a multi-client SSH server.

### Functions

| | |
|---|---|
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *entry_points*(**params) | Return EntryPoint objects for all installed packages. |
| *get_event_loop*() | Return an asyncio event loop. |
| *setup_logs*([config]) | Configure the logger for euporie. |

### euporie.hub.app.add_setting

euporie.hub.app.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.hub.app.entry_points

euporie.hub.app.**entry_points**(***params*) → EntryPoints

Return EntryPoint objects for all installed packages.

Pass selection parameters (group or name) to filter the result to entry points matching those properties (see EntryPoints.select()).

**Returns**

EntryPoints for all installed packages.

### euporie.hub.app.get_event_loop

euporie.hub.app.**get_event_loop**()

Return an asyncio event loop.

When called from a coroutine or a callback (e.g. scheduled with call_soon or similar API), this function will always return the running event loop.

If there is no running event loop set, the function will return the result of *get_event_loop_policy().get_event_loop()* call.

### euporie.hub.app.setup_logs

euporie.hub.app.**setup_logs**(*config:* Config | *None* = *None*) → None

Configure the logger for euporie.

### Classes

| | |
|---|---|
| *BaseApp*([title, set_title, leave_graphics, ...]) | All euporie apps. |
| *EuporieSSHServer*(app_cls) | Launch euporie hub, which serves a euporie app over SSH. |
| *HubApp*([title, set_title, leave_graphics, ...]) | Hub App. |
| *PromptToolkitSSHSession*(interact, *, enable_cpr) | |
| *UPath*(*args[, protocol]) | |

### euporie.hub.app.BaseApp

**class** euporie.hub.app.**BaseApp**(*title: str | None = None*, *set_title: bool = True*, *leave_graphics: FilterOrBool = True*, *extend_renderer_height: FilterOrBool = False*, *extend_renderer_width: FilterOrBool = False*, *enable_page_navigation_bindings: FilterOrBool | None = True*, ***kwargs: Any*)

All euporie apps.

The base euporie application class.

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

### euporie.hub.app.EuporieSSHServer

**class** euporie.hub.app.**EuporieSSHServer**(*app_cls: type[*euporie.core.app.BaseApp*]*)

Launch euporie hub, which serves a euporie app over SSH.

Launch euporie hub, a multi-client SSH server running euporie, which allows multiple users to connect and run instances of a euporie app.

### euporie.hub.app.HubApp

**class** euporie.hub.app.**HubApp**(*title: str | None = None*, *set_title: bool = True*, *leave_graphics: FilterOrBool = True*, *extend_renderer_height: FilterOrBool = False*, *extend_renderer_width: FilterOrBool = False*, *enable_page_navigation_bindings: FilterOrBool | None = True*, ***kwargs: Any*)

Hub App.

An app which runs as a multi-user SSH server.

This app never actually gets run, but is used to run another app in an SSH server.

### euporie.hub.app.PromptToolkitSSHSession

**class** euporie.hub.app.**PromptToolkitSSHSession**(*interact: Callable[[PromptToolkitSSHSession], Coroutine[Any, Any, None]]*, ***, *enable_cpr: bool*)

### euporie.hub.app.UPath

**class** euporie.hub.app.**UPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

**class** euporie.hub.app.**EuporieSSHServer**(*app_cls: type[euporie.core.app.BaseApp]*)

Bases: SSHServer

Launch euporie hub, which serves a euporie app over SSH.

Launch euporie hub, a multi-client SSH server running euporie, which allows multiple users to connect and run instances of a euporie app.

**auth_completed**() → None

Authentication was completed successfully

This method is called when authentication has completed succesfully. Applications may use this method to perform processing based on the authenticated username or options in the authorized keys list or certificate associated with the user before any sessions are opened or forwarding requests are handled.

**begin_auth**(*username: str*) → bool | Awaitable[bool]

Perform authentication in the SSH server.

**change_password**(*username: str*, *old_password: str*, *new_password: str*) → Union[bool, Awaitable[bool]]

Handle a request to change a user's password

This method is called when a user makes a request to change their password. It should first validate that the old password provided is correct and then attempt to change the user's password to the new value.

If the old password provided is valid and the change to the new password is successful, this method should return *True*. If the old password is not valid or password changes are not supported, it should return *False*. It may also raise PasswordChangeRequired to request that the client try again if the new password is not acceptable for some reason.

If blocking operations need to be performed to determine the validity of the old password or to change to the new password, this method may be defined as a coroutine.

By default, this method returns *False*, rejecting all password changes.

**Parameters**

- **username** (*str*) – The user whose password should be changed
- **old_password** (*str*) – The user's current password
- **new_password** (*str*) – The new password being requested

**Returns**
A *bool* indicating if the password change is successful or not

**Raises**
`PasswordChangeRequired` if the new password is not acceptable and the client should be asked to provide another

**connection_lost**(*exc: Optional[Exception]*) → None

Called when a connection is lost or closed

This method is called when a connection is closed. If the connection is shut down cleanly, *exc* will be *None*. Otherwise, it will be an exception explaining the reason for the disconnect.

**connection_made**(*conn: SSHServerConnection*) → None

Called when a connection is made

This method is called when a new TCP connection is accepted. The *conn* parameter should be stored if needed for later use.

**Parameters**
**conn** (`SSHServerConnection`) – The connection which was successfully opened

**connection_requested**(*dest_host: str*, *dest_port: int*, *orig_host: str*, *orig_port: int*) → Union[bool, SSHTCPSession, Callable[[SSHReader, SSHWriter], Union[None, Awaitable[None]]], Tuple[SSHTCPChannel, SSHTCPSession], Tuple[SSHTCPChannel, Callable[[SSHReader, SSHWriter], Union[None, Awaitable[None]]]]]

Handle a direct TCP/IP connection request

This method is called when a direct TCP/IP connection request is received by the server. Applications wishing to accept such connections must override this method.

To allow standard port forwarding of data on the connection to the requested destination host and port, this method should return *True*.

To reject this request, this method should return *False* to send back a "Connection refused" response or raise an `ChannelOpenError` exception with the reason for the failure.

If the application wishes to process the data on the connection itself, this method should return either an `SSHTCPSession` object which can be used to process the data received on the channel or a tuple consisting of of an `SSHTCPChannel` object created with `create_tcp_channel()` and an `SSHTCPSession`, if the application wishes to pass non-default arguments when creating the channel.

If blocking operations need to be performed before the session can be created, a coroutine which returns an `SSHTCPSession` object can be returned instead of the session iself. This can be either returned directly or as a part of a tuple with an `SSHTCPChannel` object.

By default, all connection requests are rejected.

**Parameters**

- **dest_host** (*str*) – The address the client wishes to connect to
- **dest_port** (*int*) – The port the client wishes to connect to
- **orig_host** (*str*) – The address the connection was originated from

- **orig_port** (*int*) – The port the connection was originated from

**Returns**

One of the following:

- An `SSHTCPSession` object or a coroutine which returns an `SSHTCPSession`
- A tuple consisting of an `SSHTCPChannel` and the above
- A *callable* or coroutine handler function which takes AsyncSSH stream objects for reading from and writing to the connection
- A tuple consisting of an `SSHTCPChannel` and the above
- *True* to request standard port forwarding
- *False* to refuse the connection

**Raises**

`ChannelOpenError` if the connection shouldn't be accepted

**debug_msg_received**(*msg: str*, *lang: str*, *always_display: bool*) → None

A debug message was received on this connection

This method is called when the other end of the connection sends a debug message. Applications should implement this method if they wish to process these debug messages.

**Parameters**

- **msg** (*str*) – The debug message sent
- **lang** (*str*) – The language the message is in
- **always_display** (*bool*) – Whether or not to display the message

**get_kbdint_challenge**(*username: str*, *lang: str*, *submethods: str*) → Union[bool, Tuple[str, str, str, Sequence[Tuple[str, bool]]], Awaitable[Union[bool, Tuple[str, str, str, Sequence[Tuple[str, bool]]]]]]

Return a keyboard-interactive auth challenge

This method should return *True* if authentication should succeed without any challenge, *False* if authentication should fail without any challenge, or an auth challenge consisting of a challenge name, instructions, a language tag, and a list of tuples containing prompt strings and booleans indicating whether input should be echoed when a value is entered for that prompt.

If blocking operations need to be performed to determine the challenge to issue, this method may be defined as a coroutine.

**Parameters**

- **username** (*str*) – The user being authenticated
- **lang** (*str*) – The language requested by the client for the challenge
- **submethods** (*str*) – A comma-separated list of the types of challenges the client can support, or the empty string if the server should choose

**Returns**

An authentication challenge as described above

**host_based_auth_supported**() → bool

Return whether or not host-based authentication is supported

This method should return *True* if client host-based authentication is supported. Applications wishing to support it must have this method return *True* and implement *validate_host_public_key()* and/or

*validate_host_ca_key()* to return whether or not the key provided by the client is valid for the client host being authenticated.

By default, it returns *False* indicating the client host based authentication is not supported.

**Returns**
A *bool* indicating if host-based authentication is supported or not

**kbdint_auth_supported**() → bool

Return whether or not keyboard-interactive authentication is supported

This method should return *True* if keyboard-interactive authentication is supported. Applications wishing to support it must have this method return *True* and implement *get_kbdint_challenge()* and *validate_kbdint_response()* to generate the apporiate challenges and validate the responses for the user being authenticated.

By default, this method returns *NotImplemented* tying this authentication to password authentication. If the application implements password authentication and this method is not overridden, keyboard-interactive authentication will be supported by prompting for a password and passing that to the password authentication callbacks.

**Returns**
A *bool* indicating if keyboard-interactive authentication is supported or not

**password_auth_supported**() → bool

Return whether or not password authentication is supported

This method should return *True* if password authentication is supported. Applications wishing to support it must have this method return *True* and implement *validate_password()* to return whether or not the password provided by the client is valid for the user being authenticated.

By default, this method returns *False* indicating that password authentication is not supported.

**Returns**
A *bool* indicating if password authentication is supported or not

**public_key_auth_supported**() → bool

Return whether or not public key authentication is supported

This method should return *True* if client public key authentication is supported. Applications wishing to support it must have this method return *True* and implement *validate_public_key()* and/or *validate_ca_key()* to return whether or not the key provided by the client is valid for the user being authenticated.

By default, it returns *False* indicating the client public key authentication is not supported.

**Returns**
A *bool* indicating if public key authentication is supported or not

**server_requested**(*listen_host: str*, *listen_port: int*) → Union[bool, SSHListener, Awaitable[Union[bool, SSHListener]]]

Handle a request to listen on a TCP/IP address and port

This method is called when a client makes a request to listen on an address and port for incoming TCP connections. The port to listen on may be *0* to request a dynamically allocated port. Applications wishing to allow TCP/IP connection forwarding must override this method.

To set up standard port forwarding of connections received on this address and port, this method should return *True*.

If the application wishes to manage listening for incoming connections itself, this method should return an `SSHListener` object that listens for new connections and calls `create_connection` on each of them to forward them back to the client or return *None* if the listener can't be set up.

If blocking operations need to be performed to set up the listener, a coroutine which returns an `SSHListener` can be returned instead of the listener itself.

To reject this request, this method should return *False*.

By default, this method rejects all server requests.

**Parameters**

- **`listen_host`** (*str*) – The address the server should listen on
- **`listen_port`** (*int*) – The port the server should listen on, or the value *0* to request that the server dynamically allocate a port

**Returns**

One of the following:

- An `SSHListener` object
- *True* to set up standard port forwarding
- *False* to reject the request
- A coroutine object which returns one of the above

**`session_requested`**`()` → *PromptToolkitSSHSession*

Return an SSH session.

**`unix_connection_requested`**(*dest_path: str*) → Union[bool, SSHUNIXSession, Callable[[SSHReader, SSHWriter], Union[None, Awaitable[None]]], Tuple[SSHUNIXChannel, SSHUNIXSession], Tuple[SSHUNIXChannel, Callable[[SSHReader, SSHWriter], Union[None, Awaitable[None]]]]]

Handle a direct UNIX domain socket connection request

This method is called when a direct UNIX domain socket connection request is received by the server. Applications wishing to accept such connections must override this method.

To allow standard path forwarding of data on the connection to the requested destination path, this method should return *True*.

To reject this request, this method should return *False* to send back a "Connection refused" response or raise an `ChannelOpenError` exception with the reason for the failure.

If the application wishes to process the data on the connection itself, this method should return either an `SSHUNIXSession` object which can be used to process the data received on the channel or a tuple consisting of of an `SSHUNIXChannel` object created with `create_unix_channel()` and an `SSHUNIXSession`, if the application wishes to pass non-default arguments when creating the channel.

If blocking operations need to be performed before the session can be created, a coroutine which returns an `SSHUNIXSession` object can be returned instead of the session iself. This can be either returned directly or as a part of a tuple with an `SSHUNIXChannel` object.

By default, all connection requests are rejected.

**Parameters**

**`dest_path`** (*str*) – The path the client wishes to connect to

**Returns**

One of the following:

- An SSHUNIXSession object or a coroutine which returns an SSHUNIXSession
- A tuple consisting of an SSHUNIXChannel and the above
- A *callable* or coroutine handler function which takes AsyncSSH stream objects for reading from and writing to the connection
- A tuple consisting of an SSHUNIXChannel and the above
- *True* to request standard path forwarding
- *False* to refuse the connection

**Raises**
ChannelOpenError if the connection shouldn't be accepted

**unix_server_requested**(*listen_path: str*) → Union[bool, SSHListener, Awaitable[Union[bool, SSHListener]]]

Handle a request to listen on a UNIX domain socket

This method is called when a client makes a request to listen on a path for incoming UNIX domain socket connections. Applications wishing to allow UNIX domain socket forwarding must override this method.

To set up standard path forwarding of connections received on this path, this method should return *True*.

If the application wishes to manage listening for incoming connections itself, this method should return an SSHListener object that listens for new connections and calls create_unix_connection on each of them to forward them back to the client or return *None* if the listener can't be set up.

If blocking operations need to be performed to set up the listener, a coroutine which returns an SSHListener can be returned instead of the listener itself.

To reject this request, this method should return *False*.

By default, this method rejects all server requests.

**Parameters**
**listen_path** (*str*) – The path the server should listen on

**Returns**

One of the following:

- An SSHListener object or a coroutine which returns an SSHListener or *False* if the listener can't be opened
- *True* to set up standard path forwarding
- *False* to reject the request

**validate_ca_key**(*username: str*, *key: SSHKey*) → Union[bool, Awaitable[bool]]

Return whether key is an authorized CA key for this user

Certificate based client authentication can be supported by passing authorized CA keys in the *authorized_client_keys* argument of create_server(), or by calling set_authorized_keys on the server connection from the *begin_auth()* method. However, for more flexibility in matching on the allowed set of keys, this method can be implemented by the application to do the matching itself. It should return *True* if the specified key is a valid certificate authority key for the user being authenticated.

This method may be called multiple times with different keys provided by the client. Applications should precompute as much as possible in the *begin_auth()* method so that this function can quickly return whether the key provided is in the list.

If blocking operations need to be performed to determine the validity of the key, this method may be defined as a coroutine.

By default, this method returns *False* for all CA keys.

---

**Note:** This function only needs to report whether the public key provided is a valid CA key for this user. If it is, AsyncSSH will verify that the certificate is valid, that the user is one of the valid principals for the certificate, and that the client possesses the private key corresponding to the public key in the certificate before allowing the authentication to succeed.

---

**Parameters**

- **username** (*str*) – The user being authenticated
- **key** (`SSHKey` *public key*) – The public key which signed the certificate sent by the client

**Returns**
A *bool* indicating if the specified key is a valid CA key for the user being authenticated

**validate_gss_principal** (*username: str*, *user_principal: str*, *host_principal: str*) → Union[bool, Awaitable[bool]]

Return whether a GSS principal is valid for this user

This method should return *True* if the specified user principal is valid for the user being authenticated. It can be overridden by applications wishing to perform their own authentication.

If blocking operations need to be performed to determine the validity of the principal, this method may be defined as a coroutine.

By default, this method will return *True* only when the name in the user principal exactly matches the username and the domain of the user principal matches the domain of the host principal.

**Parameters**

- **username** (*str*) – The user being authenticated
- **user_principal** (*str*) – The user principal sent by the client
- **host_principal** (*str*) – The host principal sent by the server

**Returns**
A *bool* indicating if the specified user principal is valid for the user being authenticated

**validate_host_based_user** (*username: str*, *client_host: str*, *client_username: str*) → Union[bool, Awaitable[bool]]

Return whether remote host and user is authorized for this user

This method should return *True* if the specified client host and user is valid for the user being authenticated. It can be overridden by applications wishing to enforce restrictions on which remote users are allowed to authenticate as particular local users.

If blocking operations need to be performed to determine the validity of the client host and user, this method may be defined as a coroutine.

By default, this method will return *True* when the client username matches the name of the user being authenticated.

**Parameters**

- **username** (*str*) – The user being authenticated
- **client_host** (*str*) – The hostname of the client host making the request
- **client_username** (*str*) – The username of the user on the client host

**Returns**
A *bool* indicating if the specified client host and user is valid for the user being authenticated

**validate_host_ca_key**(*client_host: str*, *client_addr: str*, *client_port: int*, *key: SSHKey*) → bool

Return whether key is an authorized CA key for this client host

Certificate based client host authentication can be supported by passing authorized host CA keys in the *known_client_hosts* argument of `create_server()`. However, for more flexibility in matching on the allowed set of keys, this method can be implemented by the application to do the matching itself. It should return *True* if the specified key is a valid certificate authority key for the client host being authenticated.

This method may be called multiple times with different keys provided by the client. Applications should precompute as much as possible in the *begin_auth()* method so that this function can quickly return whether the key provided is in the list.

By default, this method returns *False* for all CA keys.

---

**Note:** This function only needs to report whether the public key provided is a valid CA key for this client host. If it is, AsyncSSH will verify that the certificate is valid, that the client host is one of the valid principals for the certificate, and that the client possesses the private key corresponding to the public key in the certificate before allowing the authentication to succeed.

---

**Parameters**

- **client_host** (*str*) – The hostname of the client host
- **client_addr** (*str*) – The IP address of the client host
- **client_port** (*int*) – The port number on the client host
- **key** (`SSHKey` *public key*) – The public key which signed the certificate sent by the client

**Returns**
A *bool* indicating if the specified key is a valid CA key for the client host being authenticated

**validate_host_public_key**(*client_host: str*, *client_addr: str*, *client_port: int*, *key: SSHKey*) → bool

Return whether key is an authorized host key for this client host

Host key based client authentication can be supported by passing authorized host keys in the *known_client_hosts* argument of `create_server()`. However, for more flexibility in matching on the allowed set of keys, this method can be implemented by the application to do the matching itself. It should return *True* if the specified key is a valid host key for the client host being authenticated.

This method may be called multiple times with different keys provided by the client. Applications should precompute as much as possible in the *begin_auth()* method so that this function can quickly return whether the key provided is in the list.

By default, this method returns *False* for all client host keys.

---

**Note:** This function only needs to report whether the public key provided is a valid key for this client host. If it is, AsyncSSH will verify that the client possesses the corresponding private key before allowing the authentication to succeed.

---

**Parameters**

- **client_host** (*str*) – The hostname of the client host

- **client_addr** (*str*) – The IP address of the client host
- **client_port** (*int*) – The port number on the client host
- **key** (SSHKey *public key*) – The host public key sent by the client

**Returns**
A *bool* indicating if the specified key is a valid key for the client host being authenticated

**validate_kbdint_response** (*username: str*, *responses: Sequence[str]*) → Union[bool, Tuple[str, str, str, Sequence[Tuple[str, bool]]], Awaitable[Union[bool, Tuple[str, str, str, Sequence[Tuple[str, bool]]]]]]

Return whether the keyboard-interactive response is valid for this user

This method should validate the keyboard-interactive responses provided and return *True* if authentication should succeed with no further challenge, *False* if authentication should fail, or an additional auth challenge in the same format returned by *get_kbdint_challenge()*. Any series of challenges can be returned this way. To print a message in the middle of a sequence of challenges without prompting for additional data, a challenge can be returned with an empty list of prompts. After the client acknowledges this message, this function will be called again with an empty list of responses to continue the authentication.

If blocking operations need to be performed to determine the validity of the response or the next challenge to issue, this method may be defined as a coroutine.

**Parameters**

- **username** (*str*) – The user being authenticated
- **responses** (*list* of *str*) – A list of responses to the last challenge

**Returns**
*True*, *False*, or the next challenge

**validate_password** (*username: str*, *password: str*) → Union[bool, Awaitable[bool]]

Return whether password is valid for this user

This method should return *True* if the specified password is a valid password for the user being authenticated. It must be overridden by applications wishing to support password authentication.

If the password provided is valid but expired, this method may raise PasswordChangeRequired to request that the client provide a new password before authentication is allowed to complete. In this case, the application must override *change_password()* to handle the password change request.

This method may be called multiple times with different passwords provided by the client. Applications may wish to limit the number of attempts which are allowed. This can be done by having *password_auth_supported()* begin returning *False* after the maximum number of attempts is exceeded.

If blocking operations need to be performed to determine the validity of the password, this method may be defined as a coroutine.

By default, this method returns *False* for all passwords.

**Parameters**

- **username** (*str*) – The user being authenticated
- **password** (*str*) – The password sent by the client

**Returns**
A *bool* indicating if the specified password is valid for the user being authenticated

**Raises**
PasswordChangeRequired if the password provided is expired and needs to be changed

**validate_public_key**(*username: str*, *key: SSHKey*) → Union[bool, Awaitable[bool]]

Return whether key is an authorized client key for this user

Key based client authentication can be supported by passing authorized keys in the *authorized_client_keys* argument of `create_server()`, or by calling `set_authorized_keys` on the server connection from the *begin_auth()* method. However, for more flexibility in matching on the allowed set of keys, this method can be implemented by the application to do the matching itself. It should return *True* if the specified key is a valid client key for the user being authenticated.

This method may be called multiple times with different keys provided by the client. Applications should precompute as much as possible in the *begin_auth()* method so that this function can quickly return whether the key provided is in the list.

If blocking operations need to be performed to determine the validity of the key, this method may be defined as a coroutine.

By default, this method returns *False* for all client keys.

---

**Note:** This function only needs to report whether the public key provided is a valid client key for this user. If it is, AsyncSSH will verify that the client possesses the corresponding private key before allowing the authentication to succeed.

---

**Parameters**

- **username** (*str*) – The user being authenticated
- **key** (`SSHKey` *public key*) – The public key sent by the client

**Returns**

A *bool* indicating if the specified key is a valid client key for the user being authenticated

**class** euporie.hub.app.**HubApp**(*title: str | None = None*, *set_title: bool = True*, *leave_graphics: FilterOrBool = True*, *extend_renderer_height: FilterOrBool = False*, *extend_renderer_width: FilterOrBool = False*, *enable_page_navigation_bindings: FilterOrBool | None = True*, ***kwargs: Any*)

Bases: *BaseApp*

Hub App.

An app which runs as a multi-user SSH server.

This app never actually gets run, but is used to run another app in an SSH server.

**async cancel_and_wait_for_background_tasks**() → None

Cancel all background tasks, and wait for the cancellation to complete. 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.)

**cleanup**(*signum: int*, *frame: FrameType | None*) → None

Restore the state of the terminal on unexpected exit.

**cleanup_closed_tab**(*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:* Tab | *None* = *None*) → None

Close a notebook tab.

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

**property 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.

**color_palette: *ColorPalette***

**context: contextvars.Context | None**

**cpr_not_supported_callback**() → None

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

**create_background_task**(*coroutine: Coroutine[Any, Any, None]*) → Task[None]

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

Given that we still support Python versions before 3.11, we can't use task groups (and exception groups), because of that, these background tasks are not allowed to raise exceptions. If they do, we'll call the default exception handler from the event loop.

If at some point, we have Python 3.11 as the minimum supported Python version, then we can use a *TaskGroup* (with the lifetime of *Application.run_async()*, and run run the background tasks in there.

This is not threadsafe.

**create_merged_style**() → *BaseStyle*

Generate a new merged style for the application.

Using a dynamic style has serious performance issues, so instead we update the style on the renderer directly when it changes in *self.update_style*

**Returns**
Return a combined style to use for the application

**property current_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: SearchState**

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

**dialogs: dict[str, *Dialog*]**

**draw**(*render_as_done: bool* = *True*) → None

Draw the app without focus, leaving the cursor below the drawn output.

**exit** (*result: Optional[_AppResult] = None*, *exception: BaseException | type[BaseException] | None = 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.)

**focus_tab** (*tab:* Tab) → None

Make a tab visible and focuses it.

**focused_element: FocusableElement | None**

**formatters: list[*Formatter*]**

**full_screen: bool**

**future: Future[_AppResult] | None**

**get_edit_mode** () → *EditingMode*

Return the editing mode enum defined in the configuration.

**get_file_tab** (*path:* Path) → type[*Tab*] | None

Return the tab to use for a file path.

**get_file_tabs** (*path:* Path) → list[type[*Tab*]]

Return the tab to use for a file path.

**get_language_lsps** (*language: str*) → list[*euporie.core.lsp.LspClient*]

Return the approprrate LSP clients for a given language.

**get_used_style_strings** () → list[str]

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

**graphics: *WeakSet*[*Float*]**

**async classmethod interact** (*ssh_session:* PromptToolkitSSHSession) → None

Run the app asynchronously for the hub SSH server.

**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

Launch the HubApp SSH server.

**load_container**() → *FloatContainer*

Load the root container for this application.

**Returns**

The root container for this app

**classmethod load_input**() → Input

Create the input for this application to use.

Ensures the TUI app always tries to run in a TTY.

**Returns**

A prompt-toolkit input instance

**load_key_bindings**() → None

Load the application's key bindings.

**classmethod load_output**() → Output

Create the output for this application to use.

Ensures the TUI app always tries to run in a TTY.

**Returns**

A prompt-toolkit output instance

**log_stdout_level: str = 'CRITICAL'**

**loop: AbstractEventLoop | None**

**lsp_clients: *WeakValueDictionary*[str, *LspClient*]**

**menus: dict[str, *Float*]**

**mouse_limits: *WritePosition* | None**

**mouse_position: *Point***

**name: str = 'hub'**

**open_file**(*path:* Path, *read_only:* *bool* = *False*, *tab_class:* *type*[Tab] | *None* = *None*) → None

Create 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
- **tab_class** – The tab type to use to open the file

**open_files**() → None

Open the files defined in the configuration.

**pager**: *Pager* | None

**pause_rendering**() → None

Block rendering, but allows input to be processed.

The first line prevents the display being drawn, and the second line means the key processor continues to process keys. We need this as we need to wait for the results of terminal queries which come in as key events.

This is used to prevent flicker when we update the styles based on terminal feedback.

**post_load**() → None

Allow subclasses to define additional loading steps.

**post_load_callables**: list[Callable[[], None]]

**pre_run**(*app:* prompt_toolkit.application.application.Application | *None* = *None*) → None

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

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

**print_text**(*text: AnyFormattedText*, *style:* BaseStyle | *None* = *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.

**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.

**resume_rendering**() → None

Reume rendering the app.

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

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

This will run the application in a fresh asyncio event loop.

**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_appp().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: Callable[[], None] | None = None*, *set_exception_handler: bool = True*, *handle_sigint: bool = True*, *slow_callback_duration: float = 0.5*) → _AppResult

Run the application.

**async run_system_command**(*command: str*, *wait_for_enter: bool = True*, *display_before_text: AnyFormattedText = ''*, *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.

**search_bar: *SearchBar* | None**

**shutdown_lsps**() → None

Shut down all the remaining LSP servers.

**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 syntax_theme: str**

Calculate the current syntax theme.

**property tab: *Tab* | None**

Return the currently selected tab container object.

**property tab_idx: int**

Get the current tab index.

**tabs: list[*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 uses 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.

**property title: str**

The application's title.

**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_edit_mode**(*setting:* Setting | *None* = *None*) → None

Set the keybindings for editing mode.

**update_style**(*query:* TerminalQuery | Setting | *None* = *None*) → None

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

**vi_state**

Vi state. (For Vi key bindings.)

### 5.12.4 euporie.notebook

Define euporie's application classes.

#### Modules

| | |
|---|---|
| *euporie.notebook.app* | A text base user interface for euporie. |
| *euporie.notebook.current* | Allow access to the current running application. |
| *euporie.notebook.enums* | Define enums. |
| *euporie.notebook.filters* | Define filters used in the notebook app. |
| *euporie.notebook.tabs* | Tab for use in euporie notebook editor. |
| *euporie.notebook.widgets* | Contain widgets used in the notebook app. |

#### euporie.notebook.app

A text base user interface for euporie.

#### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *cast*(typ, val) | Cast a value to a type. |
| *get_cmd*(name) | Get a command from the centralized command system by name. |
| *register_bindings*(bindings) | Update the key-binding registry. |
| *to_formatted_text*(value[, style, auto_convert]) | Convert the given value (which can be formatted text) into a list of text fragments. |
| *truncate*(ft, width[, style, placeholder, ...]) | Truncate all lines at a given length. |

### euporie.notebook.app.add_cmd

euporie.notebook.app.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.notebook.app.add_setting

euporie.notebook.app.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.notebook.app.cast

euporie.notebook.app.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.notebook.app.get_cmd

euporie.notebook.app.**get_cmd**(*name: str*) → *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.notebook.app.register_bindings

euporie.notebook.app.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### euporie.notebook.app.to_formatted_text

euporie.notebook.app.**to_formatted_text**(*value: AnyFormattedText*, *style: str = ''*, *auto_convert: bool = False*) → *FormattedText*

Convert the given value (which can be formatted text) into a list of text fragments. (Which is the canonical form of formatted text.) The outcome is always a *FormattedText* instance, which is a list of (style, text) tuples.

It can take a plain text string, an *HTML* or *ANSI* object, anything that implements *__pt_formatted_text__* or a callable that takes no arguments and returns one of those.

**Parameters**

- **style** – An additional style string which is applied to all text fragments.
- **auto_convert** – If *True*, also accept other types, and convert them to a string first.

### euporie.notebook.app.truncate

euporie.notebook.app.**truncate**(*ft: StyleAndTextTuples*, *width: int*, *style: str = ''*, *placeholder: str = '…'*, *ignore_whitespace: bool = False*) → StyleAndTextTuples

Truncate all lines at a given length.

**Parameters**

- **ft** – The formatted text to truncate
- **width** – The width at which to truncate the text
- **style** – The style to apply to the truncation placeholder. The style of the truncated text will be used if not provided
- **placeholder** – The string that will appear at the end of a truncated line
- **ignore_whitespace** – Do not use placeholder when truncating whitespace

**Returns**

The truncated formatted text

## Classes

| | |
|---|---|
| *AboutDialog*(app) | A dialog which shows an "about" message. |
| *BaseApp*([title, set_title, leave_graphics, ...]) | All euporie apps. |
| *CommandPalette*(app) | A command palette which allows searching the available commands. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *ConfirmDialog*(app) | A dialog which allows the user to confirm an action. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *DynamicContainer*(get_container) | Container class that dynamically returns any Container. |
| *ErrorDialog*(app) | A dialog to show unhandled exceptions. |
| *FileBrowser*([path, on_select, on_open, ...]) | A file browser. |
| *FloatContainer*(content, floats[, modal, ...]) | A *FloatContainer* which uses :py`BoundedWritePosition`s. |

continues on next page

Table 11 – continued from previous page

| | |
|---|---|
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *MenuBar*(app, menu_items, grid) | A container to hold the menubar and main application body. |
| *MenuItem*([formatted_text, description, ...]) | A prompt-toolkit compatible menu item with more advanced capabilities. |
| *MsgBoxDialog*(app) | A dialog which shows the user a message. |
| *NoKernelsDialog*(app) | Dialog to warn the user that no installed kernels were found. |
| *Notebook*(app[, path, kernel, comms, ...]) | Interactive notebooks. |
| *NotebookApp*(**kwargs) | Notebook app. |
| *OpenFileDialog*(app) | A dialog which prompts the user for a filepath to open. |
| *Pager*([height]) | Interactive help pager. |
| *Pattern*(char[, pattern]) | Fill an area with a repeating background pattern. |
| *SaveAsDialog*(app) | A dialog which prompts the user for a filepath to save the current tab. |
| *SearchBar*([search_buffer, vi_mode, ...]) | Search mode. |
| *SelectKernelDialog*(app) | A dialog which allows the user to select a kernel. |
| *ShortcutsDialog*(app) | Display details of registered key-bindings in a dialog. |
| *SideBar*(titles, icons, panels) | A side-bar for the notebook application. |
| *StatusBar*([extra_filter, default]) | A status bar which shows the status of the current tab. |
| *StatusContainer*(body, status) | A container which allows attaching a status function. |
| *TabBarControl*(tabs, active[, spacing, ...]) | A control which shows a tab bar. |
| *TabBarTab*(title, on_activate[, ...]) | A named tuple represting a tab and it's callbacks. |
| *TabMode*(value[, names, module, qualname, ...]) | Define how multiple tabs are displayed. |
| *UPath*(*args[, protocol]) | |
| *UnsavedDialog*(app) | A dialog prompting the user to save unsaved changes. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *WindowAlign*(value[, names, module, ...]) | Alignment of the Window content. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.notebook.app.AboutDialog

**class** euporie.notebook.app.**AboutDialog**(*app:* BaseApp)

A dialog which shows an “about” message.

### euporie.notebook.app.BaseApp

**class** euporie.notebook.app.**BaseApp**(*title: str | None = None*, *set_title: bool = True*, *leave_graphics: FilterOrBool = True*, *extend_renderer_height: FilterOrBool = False*, *extend_renderer_width: FilterOrBool = False*, *enable_page_navigation_bindings: FilterOrBool | None = True*, ***kwargs: Any*)

All euporie apps.

The base euporie application class.

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

### euporie.notebook.app.CommandPalette

**class** euporie.notebook.app.**CommandPalette**(*app:* BaseApp)

A command palette which allows searching the available commands.

### euporie.notebook.app.Condition

**class** euporie.notebook.app.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.notebook.app.ConditionalContainer

**class** euporie.notebook.app.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.notebook.app.ConfirmDialog

**class** euporie.notebook.app.**ConfirmDialog**(*app:* BaseApp)

A dialog which allows the user to confirm an action.

### euporie.notebook.app.Dimension

**class** euporie.notebook.app.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.

- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.notebook.app.DynamicContainer

**class** euporie.notebook.app.**DynamicContainer**(*get_container: Callable[[], AnyContainer]*)

Container class that dynamically returns any Container.

**Parameters**

**get_container** – Callable that returns a *Container* instance or any widget with a `__pt_container__` method.

### euporie.notebook.app.ErrorDialog

**class** euporie.notebook.app.**ErrorDialog**(*app:* BaseApp)

A dialog to show unhandled exceptions.

### euporie.notebook.app.FileBrowser

**class** euporie.notebook.app.**FileBrowser**(*path:* Path | *None* = *None*, *on_select: Callable[[*Path*], None]* | *None* = *None*, *on_open: Callable[[*Path*], None]* | *None* = *None*, *on_chdir: Callable[[*Path*], None]* | *None* = *None*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *style: str* = *''*, *show_address_bar: FilterOrBool* = *True*)

A file browser.

### euporie.notebook.app.FloatContainer

**class** euporie.notebook.app.**FloatContainer**(*content: AnyContainer*, *floats: list[*Float*]*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*, *z_index: int* | *None* = *None*)

A *FloatContainer* which uses :py`BoundedWritePosition`s.

### euporie.notebook.app.FormattedTextControl

**class** euporie.notebook.app.**FormattedTextControl**(*text: AnyFormattedText* = *''*, *style: str* = *''*, *focusable: FilterOrBool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *show_cursor: bool* = *True*, *modal: bool* = *False*, *get_cursor_position: Callable[[],* Point | *None]* | *None* = *None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **`focusable`** – *bool* or *`Filter`*: Tell whether this control is focusable.
- **`text`** – Text or formatted text to be displayed.
- **`style`** – Style string applied to the content. (If you want to style the whole `Window`, pass the style to the `Window` instead.)
- **`key_bindings`** – a *`KeyBindings`* object.
- **`get_cursor_position`** – A callable that returns the cursor position as a *Point* instance.

### euporie.notebook.app.HSplit

**class** euporie.notebook.app.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str* | *None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int* | *None = None*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str* | *Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

### euporie.notebook.app.MenuBar

**class** euporie.notebook.app.**MenuBar**(*app: BaseApp, menu_items: Sequence[MenuItem], grid: GridStyle = □□□□ □ □□ □□□□ □□□□*)

A container to hold the menubar and main application body.

### euporie.notebook.app.MenuItem

**class** euporie.notebook.app.**MenuItem**(*formatted_text: AnyFormattedText = ''*, *description: str = ''*, *separator: bool = False*, *handler: Callable[[], None] | None = None*, *children: list[MenuItem] | None = None*, *shortcut: AnyFormattedText = ''*, *hidden: FilterOrBool = False*, *disabled: FilterOrBool = False*, *toggled: Filter | None = None*, *collapse_prefix: bool = False*, *collapse_suffix: bool = True*)

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.

### euporie.notebook.app.MsgBoxDialog

**class** euporie.notebook.app.**MsgBoxDialog**(*app: BaseApp*)

A dialog which shows the user a message.

### euporie.notebook.app.NoKernelsDialog

**class** euporie.notebook.app.**NoKernelsDialog**(*app: BaseApp*)

Dialog to warn the user that no installed kernels were found.

### euporie.notebook.app.Notebook

**class** euporie.notebook.app.**Notebook**(*app: BaseApp*, *path: Path | None = None*, *kernel: Kernel | None = None*, *comms: dict[str, Comm] | None = None*, *use_kernel_history: bool = True*, *json: dict[str, Any] | None = None*)

Interactive notebooks.

A tab which allows running and editing a notebook.

### euporie.notebook.app.NotebookApp

**class** euporie.notebook.app.**NotebookApp**(***kwargs: Any*)

Notebook app.

Interactively edit a notebook file.

Launch the interactive TUI notebook editor, allowing you to run and edit Jupyter notebooks in the terminal.

### euporie.notebook.app.OpenFileDialog

**class** euporie.notebook.app.**OpenFileDialog**(*app:* BaseApp)

A dialog which prompts the user for a filepath to open.

### euporie.notebook.app.Pager

**class** euporie.notebook.app.**Pager**(*height: AnyDimension | None = None*)

Interactive help pager.

A pager which displays information at the bottom of a tab.

### euporie.notebook.app.Pattern

**class** euporie.notebook.app.**Pattern**(*char: str | Callable[[], str]*, *pattern: int | Callable[[], int] = 1*)

Fill an area with a repeating background pattern.

### euporie.notebook.app.SaveAsDialog

**class** euporie.notebook.app.**SaveAsDialog**(*app:* BaseApp)

A dialog which prompts the user for a filepath to save the current tab.

### euporie.notebook.app.SearchBar

**class** euporie.notebook.app.**SearchBar**(*search_buffer:* Buffer | *None = None*, *vi_mode: bool = False*, *text_if_not_searching: AnyFormattedText = ''*, *forward_search_prompt: AnyFormattedText = 'I-search: '*, *backward_search_prompt: AnyFormattedText = 'I-search backward: '*, *ignore_case: FilterOrBool = False*)

Search mode.

A search toolbar with custom style and text.

### euporie.notebook.app.SelectKernelDialog

**class** euporie.notebook.app.**SelectKernelDialog**(*app:* BaseApp)

A dialog which allows the user to select a kernel.

### euporie.notebook.app.ShortcutsDialog

**class** euporie.notebook.app.**ShortcutsDialog**(*app:* BaseApp)

Display details of registered key-bindings in a dialog.

### euporie.notebook.app.SideBar

**class** euporie.notebook.app.**SideBar**(*titles: Sequence[str]*, *icons: Sequence[str]*, *panels: Sequence[AnyContainer]*)

A side-bar for the notebook application.

### euporie.notebook.app.StatusBar

**class** euporie.notebook.app.**StatusBar**(*extra_filter: FilterOrBool = True*, *default: StatusBarFields | None = None*)

A status bar which shows the status of the current tab.

### euporie.notebook.app.StatusContainer

**class** euporie.notebook.app.**StatusContainer**(*body: AnyContainer*, *status: Callable[[], StatusBarFields | None]*)

A container which allows attaching a status function.

### euporie.notebook.app.TabBarControl

**class** euporie.notebook.app.**TabBarControl**(*tabs: Sequence[TabBarTab] | Callable[[], Sequence[TabBarTab]]*, *active: int | Callable[[], int]*, *spacing: int = 1*, *closeable: bool = False*, *max_title_width: int = 30*)

A control which shows a tab bar.

### euporie.notebook.app.TabBarTab

**class** euporie.notebook.app.**TabBarTab**(*title: AnyFormattedText*, *on_activate: Callable*, *on_deactivate: Callable | None = None*, *on_close: Callable | None = None*)

A named tuple represting a tab and it's callbacks.

### euporie.notebook.app.TabMode

**class** euporie.notebook.app.**TabMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Define how multiple tabs are displayed.

### euporie.notebook.app.UPath

**class** euporie.notebook.app.**UPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

### euporie.notebook.app.UnsavedDialog

**class** euporie.notebook.app.**UnsavedDialog**(*app:* BaseApp)

A dialog prompting the user to save unsaved changes.

### euporie.notebook.app.VSplit

**class** euporie.notebook.app.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.notebook.app.Window

**class** euporie.notebook.app.**Window**(*content:* UIControl | *None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[*Margin*] | None = None*, *right_margins: Sequence[*Margin*] | None = None*, *scroll_offsets:* ScrollOffsets | *None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[*Window*], int] | None = None*, *get_horizontal_scroll: Callable[[*Window*], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align:* WindowAlign | *Callable[[],* WindowAlign*] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.notebook.app.WindowAlign

**class** euporie.notebook.app.**WindowAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of the Window content.

Note that this is different from *HorizontalAlign* and *VerticalAlign*, which are used for the alignment of the child containers in respectively *VSplit* and *HSplit*.

### euporie.notebook.app.partial

**class** euporie.notebook.app.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.notebook.app.**NotebookApp**(***kwargs: Any*)

Bases: *BaseApp*

Notebook app.

Interactively edit a notebook file.

Launch the interactive TUI notebook editor, allowing you to run and edit Jupyter notebooks in the terminal.

**async cancel_and_wait_for_background_tasks**() → None

Cancel all background tasks, and wait for the cancellation to complete. 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: *Cell* | None**

Return the currently active cell.

**cleanup**(*signum: int*, *frame: FrameType* | *None*) → None

Restore the state of the terminal on unexpected exit.

**cleanup_closed_tab**(*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:* Tab | *None* = *None*) → None

Close a notebook tab.

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

**property 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.

**color_palette: *ColorPalette***

**context: contextvars.Context | None**

**cpr_not_supported_callback**() → None

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

**create_background_task**(*coroutine: Coroutine[Any, Any, None]*) → Task[None]

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

Given that we still support Python versions before 3.11, we can't use task groups (and exception groups), because of that, these background tasks are not allowed to raise exceptions. If they do, we'll call the default exception handler from the event loop.

If at some point, we have Python 3.11 as the minimum supported Python version, then we can use a *TaskGroup* (with the lifetime of *Application.run_async()*, and run run the background tasks in there.

This is not threadsafe.

**create_merged_style**() → *BaseStyle*

Generate a new merged style for the application.

Using a dynamic style has serious performance issues, so instead we update the style on the renderer directly when it changes in *self.update_style*

**Returns**

Return a combined style to use for the application

**property current_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: SearchState**

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

**dialogs: dict[str, *Dialog*]**

**draw**(*render_as_done: bool = True*) → None

Draw the app without focus, leaving the cursor below the drawn output.

**exit**(**args: Any*, ***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**

- **args** – Positional arguments
- **kwargs** – Key word arguments

**focus_tab**(*tab:* Tab) → None

Make a tab visible and focuses it.

**focused_element: FocusableElement | None**

**format_title**() → StyleAndTextTuples
Format the tab's title for display in the top right of the app.

**formatters: list[*Formatter*]**

**full_screen: bool**

**future: Future[_AppResult] | None**

**get_edit_mode**() → *EditingMode*
Return the editing mode enum defined in the configuration.

**get_file_tab**(*path:* Path) → type[*Tab*] | None
Return the tab to use for a file path.

**get_file_tabs**(*path:* Path) → list[type[*Tab*]]
Return the tab to use for a file path.

**get_language_lsps**(*language: str*) → list[*euporie.core.lsp.LspClient*]
Return the appropriate LSP clients for a given language.

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

**graphics: *WeakSet*[*Float*]**

**async classmethod interact**(*ssh_session:* PromptToolkitSSHSession) → None
Run the app asynchronously for the hub SSH server.

**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
Launch the app.

**load_container**() → *FloatContainer*
Build the main application layout.

**classmethod load_input**() → Input
Create the input for this application to use.

Ensures the TUI app always tries to run in a TTY.

**Returns**
A prompt-toolkit input instance

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

**load_menu_items**() → list[*euporie.core.widgets.menu.MenuItem*]

Load the list of menu items to display in the menu.

**classmethod load_output**() → Output

Create the output for this application to use.

Ensures the TUI app always tries to run in a TTY.

**Returns**

A prompt-toolkit output instance

**log_stdout_level: str = 'CRITICAL'**

**loop: AbstractEventLoop | None**

**lsp_clients: *WeakValueDictionary*[str, *LspClient*]**

**menus: dict[str, *Float*]**

**mouse_limits: *WritePosition* | None**

**mouse_position: *Point***

**name: str = 'notebook'**

**property notebook: *euporie.notebook.tabs.notebook.Notebook* | None**

Return the currently active notebook.

**open_file**(*path:* Path, *read_only:* *bool* = *False*, *tab_class:* *type*[Tab] | *None* = *None*) → None

Create 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
- **tab_class** – The tab type to use to open the file

**open_files**() → None

Open the files defined in the configuration.

**pager: *Pager* | None**

**pause_rendering**() → None

Block rendering, but allows input to be processed.

The first line prevents the display being drawn, and the second line means the key processor continues to process keys. We need this as we need to wait for the results of terminal queries which come in as key events.

This is used to prevent flicker when we update the styles based on terminal feedback.

**post_load**() → None

Allow subclasses to define additional loading steps.

**post_load_callables: list[Callable[[], None]]**

**pre_run**(*app:* Application | *None* = *None*) → None

Continue loading the app.

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

**print_text**(*text: AnyFormattedText*, *style:* BaseStyle | *None* = *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.

**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.

**resume_rendering**() → None

Reume rendering the app.

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

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

This will run the application in a fresh asyncio event loop.

**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_appp().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: Callable[[], None]* | *None* = *None*, *set_exception_handler: bool = True*, *handle_sigint: bool = True*, *slow_callback_duration: float = 0.5*) → _AppResult

Run the application.

**async run_system_command**(*command: str*, *wait_for_enter: bool = True*, *display_before_text: AnyFormattedText = ''*, *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.

**search_bar: *SearchBar* | None**

**shutdown_lsps**() → None
Shut down all the remaining LSP servers.

**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 syntax_theme: str**
Calculate the current syntax theme.

**property tab: *Tab* | None**
Return the currently selected tab container object.

**tab_bar_tabs**() → list[*euporie.core.widgets.layout.TabBarTab*]
Return a list of the current tabs for the tab-bar.

**tab_container**() → AnyContainer
Return a container with all opened tabs.

**Returns**
A layout displaying the opened tab containers.

**property tab_idx: int**
Get the current tab index.

**tabs: list[*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 uses 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.

**property title: str**
The application's title.

**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_edit_mode**(*setting:* Setting | *None = None*) → None
Set the keybindings for editing mode.

**update_style**(*query:* TerminalQuery | Setting | *None* = *None*) → None

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

**vi_state**

Vi state. (For Vi key bindings.)

### euporie.notebook.current

Allow access to the current running application.

### Functions

| | |
|---|---|
| *cast*(typ, val) | Cast a value to a type. |
| *get_app*() | Get the current application. |
| *ptk_get_app*() | Get the current active (running) Application. |

### euporie.notebook.current.cast

euporie.notebook.current.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.notebook.current.get_app

euporie.notebook.current.**get_app**() → *NotebookApp*

Get the current application.

### euporie.notebook.current.ptk_get_app

euporie.notebook.current.**ptk_get_app**() → *Application*[Any]

Get the current active (running) Application. An *Application* is active during the `Application.run_async()` call.

We assume that there can only be one *Application* active at the same time. There is only one terminal window, with only one stdin and stdout. This makes the code significantly easier than passing around the *Application* everywhere.

If no *Application* is running, then return by default a `DummyApplication`. For practical reasons, we prefer to not raise an exception. This way, we don't have to check all over the place whether an actual *Application* was returned.

(For applications like pymux where we can have more than one *Application*, we'll use a work-around to handle that.)

euporie.notebook.current.**get_app**() → *NotebookApp*

Get the current application.

### euporie.notebook.enums

Define enums.

### Classes

| | |
|---|---|
| *Enum*(value[, names, module, qualname, type, ...]) | Create a collection of name/value pairs. |
| *TabMode*(value[, names, module, qualname, ...]) | Define how multiple tabs are displayed. |

### euporie.notebook.enums.Enum

**class** euporie.notebook.enums.**Enum**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes – see the documentation for details.

### euporie.notebook.enums.TabMode

**class** euporie.notebook.enums.**TabMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Define how multiple tabs are displayed.

**class** euporie.notebook.enums.**TabMode**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *Enum*

Define how multiple tabs are displayed.

**STACK = 'stack'**

**TILE_HORIZONTALLY = 'tile_horizontally'**

**TILE_VERTICALLY = 'tile_vertically'**

### euporie.notebook.filters

Define filters used in the notebook app.

### Functions

| | |
|---|---|
| *get_app*() | Get the current active (running) Application. |

### euporie.notebook.filters.get_app

euporie.notebook.filters.**get_app**() → *BaseApp*

Get the current active (running) Application.

### Classes

| | |
|---|---|
| *Condition*(func) | Turn any callable into a Filter. |

### euporie.notebook.filters.Condition

**class** euporie.notebook.filters.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.notebook.tabs

Tab for use in euporie notebook editor.

### Modules

| | |
|---|---|
| *euporie.notebook.tabs.display* | Contain a tab for displaying files. |
| *euporie.notebook.tabs.edit* | Contain a tab for displaying files. |
| *euporie.notebook.tabs.json* | Contain a tab for displaying JSON data. |
| *euporie.notebook.tabs.log* | Initiate logging for euporie.core. |
| *euporie.notebook.tabs.notebook* | Contain the main class for a notebook file. |

### euporie.notebook.tabs.display

Contain a tab for displaying files.

### Functions

| | |
|---|---|
| *get_format*(path[, default]) | Attempt to guess the format of a path. |
| *run_in_thread_with_context*(func, *args[, daemon]) | Run a function in an thread, but make sure it uses the same contextvars. |

### euporie.notebook.tabs.display.get_format

euporie.notebook.tabs.display.**get_format**(*path:* Path | *str*, *default: str* = '') → str

Attempt to guess the format of a path.

### euporie.notebook.tabs.display.run_in_thread_with_context

euporie.notebook.tabs.display.**run_in_thread_with_context**(*func: Callable*, **args: Any*, *daemon: bool* = *True*, ***kwargs: Any*) → None

Run a function in an thread, but make sure it uses the same contextvars.

This is required so that the function will see the right application.

## Classes

| | |
|---|---|
| *Datum*(data, *args, **kwargs) | Class for storing and converting display data. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *Display*(datum[, height, width, focusable, ...]) | Rich output displays. |
| *DisplayTab*(app[, path]) | Tab class for displaying files. |
| *MarginContainer*(margin, target) | A container which renders a stand-alone margin. |
| *ScrollbarMargin*([display_arrows, ...]) | Margin displaying a scrollbar. |
| *Tab*(app[, path]) | Base class for interface tabs. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |

### euporie.notebook.tabs.display.Datum

**class** euporie.notebook.tabs.display.**Datum**(*data: T*, **args: Any*, ***kwargs: Any*)

Class for storing and converting display data.

### euporie.notebook.tabs.display.Dimension

**class** euporie.notebook.tabs.display.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.notebook.tabs.display.Display

**class** euporie.notebook.tabs.display.**Display**(*datum: Datum*, *height: AnyDimension = None*, *width: AnyDimension = None*, *focusable: FilterOrBool = False*, *focus_on_click: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *always_hide_cursor: FilterOrBool = True*, *scrollbar: FilterOrBool = True*, *scrollbar_autohide: FilterOrBool = True*, *dont_extend_height: FilterOrBool = True*, *dont_extend_width: FilterOrBool = False*, *style: str | Callable[[], str] = ''*)

Rich output displays.

A container for displaying rich output data.

### euporie.notebook.tabs.display.DisplayTab

**class** euporie.notebook.tabs.display.**DisplayTab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Tab class for displaying files.

### euporie.notebook.tabs.display.MarginContainer

**class** euporie.notebook.tabs.display.**MarginContainer**(*margin:* Margin, *target:* *ScrollableContainer*)

A container which renders a stand-alone margin.

### euporie.notebook.tabs.display.ScrollbarMargin

**class** euporie.notebook.tabs.display.**ScrollbarMargin**(*display_arrows: Union[*Filter, *bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[*Filter, *bool] = False*, *smooth: bool = True*, *style: str = ''*)

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

### euporie.notebook.tabs.display.Tab

**class** euporie.notebook.tabs.display.**Tab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Base class for interface tabs.

### euporie.notebook.tabs.display.VSplit

**class** euporie.notebook.tabs.display.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align:* *HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding:* *AnyDimension = 0*, *padding_char: str* | *None* = *None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int* | *None* = *None*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

**class** euporie.notebook.tabs.display.**DisplayTab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Bases: *Tab*

Tab class for displaying files.

**close**(*cb: Callable* | *None* = *None*) → None

Close a tab with a callback.

**Parameters**

**cb** – A function to call after the tab is closed.

**container: AnyContainer**

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None

Focus the tab (or make it visible).

**load_container**() → AnyContainer

Abcract method for loading the notebook's main container.

**mime_types: ClassVar[set[str]] = {'*', 'application/pdf', 'application/x-latex', 'image/gif', 'image/jpeg', 'image/png', 'image/svg+xml', 'stream/std*', 'text/*', 'text/html', 'text/latex', 'text/markdown', 'text/x-markdown'}**

**name: str | None = 'File Viewer'**

**reset**() → None

Reset the state of the tab.

**save**(*path:* Path | *None* = *None*, *cb: Callable* | *None* = *None*) → None

Save the current notebook.

**property title: str**

Return the tab title.

**weight: int = 0**

## euporie.notebook.tabs.edit

Contain a tab for displaying files.

### Functions

| | |
|---|---|
| *detect_lexer*([text, path, language]) | Detect the pygments lexer for a file. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |

### euporie.notebook.tabs.edit.detect_lexer

euporie.notebook.tabs.edit.**detect_lexer**(*text: str = ''*, *path:* Path | *None = None*, *language: str = ''*) → PygmentsLexerCls | None

Detect the pygments lexer for a file.

### euporie.notebook.tabs.edit.load_registered_bindings

euporie.notebook.tabs.edit.**load_registered_bindings**(**names: str*, *config:* Config | *None = None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

## Classes

| | |
|---|---|
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *EditorTab*(app[, path, kernel, comms, ...]) | Tab class for editing text files. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *Kernel*(kernel_tab[, threaded, allow_stdin, ...]) | Run a notebook kernel and communicates with it asynchronously. |
| *KernelInput*(kernel_tab[, text, multiline, ...]) | Kernel input text areas. |
| *KernelTab*(app[, path, kernel, comms, ...]) | A Tab which connects to a kernel. |
| *MsgCallbacks* | Typed dictionary for named message callbacks. |
| *UntitledPath*(*args[, protocol]) | A path for untitled files, as needed for LSP servers. |

### euporie.notebook.tabs.edit.Dimension

**class** euporie.notebook.tabs.edit.**Dimension**(*min: int* | *None = None*, *max: int* | *None = None*, *weight: int* | *None = None*, *preferred: int* | *None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.notebook.tabs.edit.EditorTab

**class** euporie.notebook.tabs.edit.**EditorTab**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms:* *dict[str,* Comm*]* | *None* = *None*, *use_kernel_history:* *bool* = *False*)

Tab class for editing text files.

### euporie.notebook.tabs.edit.HSplit

**class** euporie.notebook.tabs.edit.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char:* *str* | *None* = *None*, *padding_style:* *str* = *''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index:* *int* | *None* = *None*, *modal:* *bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style:* *str* | *Callable[[],* *str]* = *''*)

Several layouts, one stacked above/under the other.

### euporie.notebook.tabs.edit.Kernel

**class** euporie.notebook.tabs.edit.**Kernel**(*kernel_tab:* KernelTab, *threaded:* *bool* = *True*, *allow_stdin:* *bool* = *False*, *default_callbacks:* MsgCallbacks | *None* = *None*, *connection_file:* Path | *None* = *None*)

Run a notebook kernel and communicates with it asynchronously.

Has the ability to run itself in it's own thread.

### euporie.notebook.tabs.edit.KernelInput

**class** euporie.notebook.tabs.edit.**KernelInput**(*kernel_tab: KernelTab*, *text: str = ''*, *multiline: FilterOrBool = True*, *password: FilterOrBool = False*, *lexer: Lexer | None = None*, *auto_suggest: AutoSuggest | None = None*, *completer: Completer | None = None*, *complete_while_typing: FilterOrBool = True*, *validator: Validator | None = None*, *accept_handler: BufferAcceptHandler | None = None*, *history: History | None = None*, *focusable: FilterOrBool = True*, *focus_on_click: FilterOrBool = True*, *wrap_lines: FilterOrBool = False*, *read_only: FilterOrBool = False*, *width: AnyDimension = None*, *height: AnyDimension = None*, *dont_extend_height: FilterOrBool = False*, *dont_extend_width: FilterOrBool = False*, *line_numbers: bool = False*, *get_line_prefix: GetLinePrefixCallable | None = None*, *scrollbar: FilterOrBool = True*, *style: str = 'class:kernel-input'*, *search_field: SearchToolbar | None = None*, *preview_search: FilterOrBool = False*, *prompt: AnyFormattedText = ''*, *input_processors: list[Processor] | None = None*, *name: str = ''*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *on_text_changed: Callable[[Buffer], None] | None = None*, *on_cursor_position_changed: Callable[[Buffer], None] | None = None*, *tempfile_suffix: str | Callable[[], str] = ''*, *key_bindings: KeyBindingsBase | None = None*, *enable_history_search: FilterOrBool = False*, *autosuggest_while_typing: FilterOrBool = True*, *validate_while_typing: FilterOrBool = False*, *scroll_offsets: ScrollOffsets | None = None*, *formatters: list[Formatter] | None = None*, *language: str | Callable[[], str] | None = None*, *diagnostics: Report | Callable[[], Report] | None = None*, *inspector: Inspector | None = None*, *show_diagnostics: FilterOrBool = True*)

Kernel input text areas.

A customized text area for the cell input.

### euporie.notebook.tabs.edit.KernelTab

**class** euporie.notebook.tabs.edit.**KernelTab**(*app: BaseApp*, *path: Path | None = None*, *kernel: Kernel | None = None*, *comms: dict[str, Comm] | None = None*, *use_kernel_history: bool = False*, *connection_file: Path | None = None*)

A Tab which connects to a kernel.

### euporie.notebook.tabs.edit.MsgCallbacks

**class** euporie.notebook.tabs.edit.**MsgCallbacks**

Typed dictionary for named message callbacks.

### euporie.notebook.tabs.edit.UntitledPath

**class** euporie.notebook.tabs.edit.**UntitledPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

A path for untitled files, as needed for LSP servers.

**class** euporie.notebook.tabs.edit.**EditorTab**(*app:* BaseApp, *path:* Path | *None = None*, *kernel:* Kernel | *None = None*, *comms: dict[str,* Comm*] | None = None*, *use_kernel_history: bool = False*)

Bases: *KernelTab*

Tab class for editing text files.

**allow_stdin: bool = True**

**bg_init = True**

**change_kernel**(*msg: str | None = None*, *startup: bool = False*) → None

Prompt the user to select a new kernel.

**close**(*cb: Callable | None = None*) → None

Check if the user want to save an unsaved notebook, then close the file.

**Parameters**

**cb** – A callback to run if after closing the notebook.

**comm_close**(*content: dict*, *buffers: Sequence[bytes]*) → None

Close a notebook Comm.

**comm_msg**(*content: dict*, *buffers: Sequence[bytes]*) → None

Respond to a Comm message from the kernel.

**comm_open**(*content: dict*, *buffers: Sequence[bytes]*) → None

Register a new kernel Comm object in the notebook.

**comms: dict[str, *Comm*]**

**completers: list[*Completer*]**

**container: AnyContainer**

**property current_input: *KernelInput***

Return the currently active kernel input, if any.

**default_callbacks: *MsgCallbacks***

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None

Focus the tab (or make it visible).

**formatters**: list[*Formatter*]

**history**: *History*

**init_kernel**(*kernel:* Kernel | *None* = *None*, *comms:* *dict[str,* Comm*]* | *None* = *None*, *use_kernel_history:* *bool* = *False*, *connection_file:* Path | *None* = *None*) → None

Set up the tab's kernel and related components.

**inspectors**: list[*Inspector*]

**interrupt_kernel**() → None

Interrupt the current *Notebook*'s kernel.

**kernel**: *Kernel*

**property kernel_display_name**: str

Return the display name of the kernel defined in the notebook JSON.

**property kernel_lang_file_ext**: str

Return the display name of the kernel defined in the notebook JSON.

**kernel_language**: str

**property kernel_name**: str

Return the name of the kernel defined in the notebook JSON.

**kernel_started**(*result:* *dict[str, Any]* | *None* = *None*) → None

Task to run when the kernel has started.

**property language**: str

Return the name of the kernel defined in the notebook JSON.

**load**() → None

Load the text file.

**load_container**() → AnyContainer

Load the "tab"'s main container.

**async load_history**() → None

Load kernel history.

**async load_lsps**() → None

Load the LSP clients.

**lsp_after_save_handler**(*lsp:* LspClient) → None

Tell the the LSP we saved a document.

**lsp_before_save_handler**(*lsp:* LspClient) → None

Tell the the LSP we are about to save a document.

**lsp_change_handler**(*lsp:* LspClient) → None

Tell the LSP server a file has changed.

**lsp_close_handler**(*lsp:* LspClient) → None

Tell the LSP we opened a file.

**lsp_open_handler**(*lsp:* LspClient) → None

Tell the LSP we opened a file.

**lsp_update_diagnostics**(*lsp:* LspClient) → None
Process a new diagnostic report from the LSP.

**lsps: list[*LspClient*]**

**property metadata: dict[str, Any]**
Return a dictionary to hold notebook / kernel metadata.

**mime_types: ClassVar[set[str]] = {'text/*'}**

**name: str | None = 'Text Editor'**

**property path_name: str**
Return the path name.

**property position: str**
Return the position of the cursor in the document.

**post_init_kernel**() → None
Load UI and file in background after kernel has inited.

**pre_init_kernel**() → None
Run stuff before the kernel is loaded.

**report**() → *Report*
Return the current diagnostic reports.

**report_kernel_error**(*error: Exception | None*) → None
Report a kernel error to the user.

**reports: *WeakKeyDictionary*[*LspClient*, *Report*]**

**reset**() → None
Reset the state of the tab.

**restart_kernel**(*cb: Callable | None = None*) → None
Restart the current *Notebook*'s kernel.

**save**(*path:* Path | *None = None*, *cb: Callable | None = None*) → None
Save the current file.

**set_kernel_info**(*info: dict*) → None
Handle kernel info requests.

**suggester: *AutoSuggest***

**property title: str**
Return the tab title.

**weight: int = 1**

### euporie.notebook.tabs.json

Contain a tab for displaying JSON data.

#### Functions

| | |
|---|---|
| *run_in_thread_with_context*(func, *args[, daemon]) | Run a function in an thread, but make sure it uses the same contextvars. |

### euporie.notebook.tabs.json.run_in_thread_with_context

euporie.notebook.tabs.json.**run_in_thread_with_context**(*func: Callable*, **args: Any*, *daemon: bool = True*, ***kwargs: Any*) → None

Run a function in an thread, but make sure it uses the same contextvars.

This is required so that the function will see the right application.

#### Classes

| | |
|---|---|
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *JsonTab*(app[, path]) | Tab class for JSON data. |
| *JsonView*(data[, title, expanded]) | A JSON-view container. |
| *Tab*(app[, path]) | Base class for interface tabs. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |

### euporie.notebook.tabs.json.Dimension

**class** euporie.notebook.tabs.json.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.notebook.tabs.json.JsonTab

**class** euporie.notebook.tabs.json.**JsonTab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Tab class for JSON data.

### euporie.notebook.tabs.json.JsonView

**class** euporie.notebook.tabs.json.**JsonView**(*data: Any*, *title: str* | *None* = *None*, *expanded: bool* = *True*)

A JSON-view container.

### euporie.notebook.tabs.json.Tab

**class** euporie.notebook.tabs.json.**Tab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Base class for interface tabs.

### euporie.notebook.tabs.json.VSplit

**class** euporie.notebook.tabs.json.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: HorizontalAlign* = *HorizontalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked left/right of the other.

**class** euporie.notebook.tabs.json.**JsonTab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Bases: *Tab*

Tab class for JSON data.

**close**(*cb: Callable* | *None* = *None*) → None

Close a tab with a callback.

**Parameters**

**cb** – A function to call after the tab is closed.

**container: AnyContainer**

**file_extensions: ClassVar[dict[str, None]] = {}**

**filte_types: ClassVar[set[str]] = {'.json'}**

**focus**() → None

Focus the tab (or make it visible).

**load_container**() → AnyContainer

Abcract method for loading the notebook's main container.

**mime_types: ClassVar[set[str]] = {'*json'}**

**name**: str | None = 'JSON Viewer'

**reset**() → None

Reset the state of the tab.

**save**(*path:* Path | *None* = *None*, *cb: Callable* | *None* = *None*) → None

Save the current notebook.

**property title**: str

Return the tab title.

**weight**: int = 0

### euporie.notebook.tabs.log

Initiate logging for euporie.core.

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *get_app*() | Get the current active (running) Application. |
| *has_focus*(value) | Enable when this buffer has the focus. |
| *parse_path*(path[, resolve]) | Parse and resolve a path. |

### euporie.notebook.tabs.log.add_cmd

euporie.notebook.tabs.log.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.notebook.tabs.log.get_app

euporie.notebook.tabs.log.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.notebook.tabs.log.has_focus

euporie.notebook.tabs.log.**has_focus**(*value: FocusableElement*) → *Condition*

Enable when this buffer has the focus.

### euporie.notebook.tabs.log.parse_path

euporie.notebook.tabs.log.**parse_path**(*path: str | PathLike*, *resolve: bool = True*) → *Path*

Parse and resolve a path.

### Classes

| | |
|---|---|
| *Condition*(func) | Turn any callable into a Filter. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *FormattedTextArea*(formatted_text, *args[, ...]) | Apply formatted text to a TextArea. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *LogView*(app[, path]) | A tab which allows you to view log entries. |
| *MarginContainer*(margin, target) | A container which renders a stand-alone margin. |
| *QueueHandler*(*args, queue[, style]) | This handler store logs events into a queue. |
| *ScrollbarMargin*([display_arrows, ...]) | Margin displaying a scrollbar. |
| *SearchToolbar*([search_buffer, vi_mode, ...]) | **param vi_mode**<br>Display '/' and '?' instead of I-search. |
| *Tab*(app[, path]) | Base class for interface tabs. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |

### euporie.notebook.tabs.log.Condition

**class** euporie.notebook.tabs.log.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.notebook.tabs.log.Dimension

**class** euporie.notebook.tabs.log.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.

- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.notebook.tabs.log.FormattedTextArea

**class** euporie.notebook.tabs.log.**FormattedTextArea**(*formatted_text: AnyFormattedText*, **args: Any*, *line_numbers: FilterOrBool = False*, ***kwargs: Any*)

Apply formatted text to a TextArea.

### euporie.notebook.tabs.log.HSplit

**class** euporie.notebook.tabs.log.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: VerticalAlign = VerticalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked above/under the other.

### euporie.notebook.tabs.log.LogView

**class** euporie.notebook.tabs.log.**LogView**(*app: BaseApp*, *path: Path | None = None*)

A tab which allows you to view log entries.

### euporie.notebook.tabs.log.MarginContainer

**class** euporie.notebook.tabs.log.**MarginContainer**(*margin: Margin*, *target: ScrollableContainer*)

A container which renders a stand-alone margin.

### euporie.notebook.tabs.log.QueueHandler

**class** euporie.notebook.tabs.log.**QueueHandler**(**args: Any*, *queue: deque*, *style: Style | None = None*, ***kwargs: Any*)

This handler store logs events into a queue.

### euporie.notebook.tabs.log.ScrollbarMargin

**class** euporie.notebook.tabs.log.**ScrollbarMargin**(*display_arrows: Union[Filter, bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[Filter, bool] = False*, *smooth: bool = True*, *style: str = ''*)

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

### euporie.notebook.tabs.log.SearchToolbar

**class** euporie.notebook.tabs.log.**SearchToolbar**(*search_buffer: Buffer | None = None*, *vi_mode: bool = False*, *text_if_not_searching: AnyFormattedText = ''*, *forward_search_prompt: AnyFormattedText = 'I-search: '*, *backward_search_prompt: AnyFormattedText = 'I-search backward: '*, *ignore_case: FilterOrBool = False*)

**Parameters**

- **vi_mode** – Display '/' and '?' instead of I-search.
- **ignore_case** – Search case insensitive.

### euporie.notebook.tabs.log.Tab

**class** euporie.notebook.tabs.log.**Tab**(*app: BaseApp*, *path: Path | None = None*)

Base class for interface tabs.

### euporie.notebook.tabs.log.VSplit

**class** euporie.notebook.tabs.log.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

**class** euporie.notebook.tabs.log.**LogView**(*app:* BaseApp, *path:* Path | *None* = *None*)

Bases: *Tab*

A tab which allows you to view log entries.

**add_record**(*message:* FormattedText) → None

Add a single new record to the textarea.

**Parameters**

**message** – The formatted log record to add

**close**(*cb: Callable* | *None* = *None*) → None

Remove log queue handler hook on close.

**container: AnyContainer**

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None

Focus the tab (or make it visible).

**mime_types: ClassVar[set[str]] = {}**

**name: str | None = None**

**reset**() → None

Reset the state of the tab.

**save**(*path:* Path | *None* = *None*, *cb: Callable* | *None* = *None*) → None

Save the current notebook.

**property title: str**

Return the title of this tab.

**weight: int = 0**

## euporie.notebook.tabs.notebook

Contain the main class for a notebook file.

## Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *deepcopy*(x[, memo, _nil]) | Deep copy operation on arbitrary Python objects. |
| *get_app*() | Get the current active (running) Application. |
| *get_cmd*(name) | Get a command from the centralized command system by name. |
| *load_registered_bindings*(*names[, config]) | Assign key-bindings to commands based on a dictionary. |
| *register_bindings*(bindings) | Update the key-binding registry. |

### euporie.notebook.tabs.notebook.add_cmd

euporie.notebook.tabs.notebook.**add_cmd**(***kwargs: Any*) → Callable

Add a command to the centralized command system.

### euporie.notebook.tabs.notebook.add_setting

euporie.notebook.tabs.notebook.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.notebook.tabs.notebook.deepcopy

euporie.notebook.tabs.notebook.**deepcopy**(*x*, *memo=None*, *_nil=[]*)

Deep copy operation on arbitrary Python objects.

See the module's __doc__ string for more info.

### euporie.notebook.tabs.notebook.get_app

euporie.notebook.tabs.notebook.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.notebook.tabs.notebook.get_cmd

euporie.notebook.tabs.notebook.**get_cmd**(*name: str*) → *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.notebook.tabs.notebook.load_registered_bindings

euporie.notebook.tabs.notebook.**load_registered_bindings**(**names: str*, *config:* Config | *None* = *None*) → *KeyBindingsBase*

Assign key-bindings to commands based on a dictionary.

### euporie.notebook.tabs.notebook.register_bindings

euporie.notebook.tabs.notebook.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### Classes

| | |
|---|---|
| *BaseNotebook*(app[, path, kernel, comms, ...]) | The main notebook container class. |
| *CachedContainer*(content[, mouse_handler_wrapper]) | A container which renders its content once and caches the output. |
| *Cell*(index, json, kernel_tab[, is_new]) | A kernel_tab cell element. |
| *ClipboardData*([text, type]) | Text on the clipboard. |
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *KernelTab*(app[, path, kernel, comms, ...]) | A Tab which connects to a kernel. |
| *Line*([char, width, height, collapse, style]) | Draw a horizontal or vertical line. |
| *MarginContainer*(margin, target) | A container which renders a stand-alone margin. |
| *MouseEventType*(value[, names, module, ...]) | |
| *Notebook*(app[, path, kernel, comms, ...]) | Interactive notebooks. |
| *Pattern*(char[, pattern]) | Fill an area with a repeating background pattern. |
| *ScrollbarMargin*([display_arrows, ...]) | Margin displaying a scrollbar. |
| *ScrollingContainer*(children[, height, ...]) | A scrollable container which renders only the currently visible children. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *deque* | deque([iterable[, maxlen]]) --> deque object |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.notebook.tabs.notebook.BaseNotebook

**class** euporie.notebook.tabs.notebook.**BaseNotebook**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *False*, *json: dict[str, Any]* | *None* = *None*)

The main notebook container class.

### euporie.notebook.tabs.notebook.CachedContainer

**class** euporie.notebook.tabs.notebook.**CachedContainer**(*content: AnyContainer*, *mouse_handler_wrapper: Callable[[MouseHandler, CachedContainer], MouseHandler] | None = None*)

A container which renders its content once and caches the output.

### euporie.notebook.tabs.notebook.Cell

**class** euporie.notebook.tabs.notebook.**Cell**(*index: int*, *json: dict*, *kernel_tab: BaseNotebook*, *is_new: bool = False*)

A kernel_tab cell element.

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

### euporie.notebook.tabs.notebook.ClipboardData

**class** euporie.notebook.tabs.notebook.**ClipboardData**(*text: str = ''*, *type: SelectionType = SelectionType.CHARACTERS*)

Text on the clipboard.

**Parameters**

- **text** – string
- **type** – *SelectionType*

### euporie.notebook.tabs.notebook.Condition

**class** euporie.notebook.tabs.notebook.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.notebook.tabs.notebook.ConditionalContainer

**class** euporie.notebook.tabs.notebook.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.notebook.tabs.notebook.Dimension

**class** euporie.notebook.tabs.notebook.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.notebook.tabs.notebook.KernelTab

**class** euporie.notebook.tabs.notebook.**KernelTab**(*app: BaseApp*, *path: Path | None = None*, *kernel: Kernel | None = None*, *comms: dict[str, Comm] | None = None*, *use_kernel_history: bool = False*, *connection_file: Path | None = None*)

A Tab which connects to a kernel.

### euporie.notebook.tabs.notebook.Line

**class** euporie.notebook.tabs.notebook.**Line**(*char: str | None = None*, *width: int | None = None*, *height: int | None = None*, *collapse: bool = False*, *style: str = 'class:grid-line'*)

Draw a horizontal or vertical line.

### euporie.notebook.tabs.notebook.MarginContainer

**class** euporie.notebook.tabs.notebook.**MarginContainer**(*margin: Margin*, *target: ScrollableContainer*)

A container which renders a stand-alone margin.

### euporie.notebook.tabs.notebook.MouseEventType

**class** euporie.notebook.tabs.notebook.**MouseEventType**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

### euporie.notebook.tabs.notebook.Notebook

**class** euporie.notebook.tabs.notebook.**Notebook**(*app: BaseApp*, *path: Path | None = None*, *kernel: Kernel | None = None*, *comms: dict[str, Comm] | None = None*, *use_kernel_history: bool = True*, *json: dict[str, Any] | None = None*)

Interactive notebooks.

A tab which allows running and editing a notebook.

### euporie.notebook.tabs.notebook.Pattern

**class** euporie.notebook.tabs.notebook.**Pattern**(*char: str | Callable[[], str]*, *pattern: int | Callable[[], int] = 1*)

Fill an area with a repeating background pattern.

### euporie.notebook.tabs.notebook.ScrollbarMargin

**class** euporie.notebook.tabs.notebook.**ScrollbarMargin**(*display_arrows: Union[Filter, bool] = True*, *up_arrow_symbol: str = '▲'*, *down_arrow_symbol: str = '▼'*, *autohide: Union[Filter, bool] = False*, *smooth: bool = True*, *style: str = ''*)

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

### euporie.notebook.tabs.notebook.ScrollingContainer

**class** euporie.notebook.tabs.notebook.**ScrollingContainer**(*children: Callable[[], Sequence[AnyContainer]] | Sequence[AnyContainer]*, *height: AnyDimension = None*, *width: AnyDimension = None*, *style: str | Callable[[], str] = ''*, *scroll_offsets: ScrollOffsets | None = None*)

A scrollable container which renders only the currently visible children.

### euporie.notebook.tabs.notebook.VSplit

**class** euporie.notebook.tabs.notebook.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.notebook.tabs.notebook.deque

**class** euporie.notebook.tabs.notebook.**deque**

deque([iterable[, maxlen]]) –> deque object

A list-like sequence optimized for data accesses near its endpoints.

### euporie.notebook.tabs.notebook.partial

**class** euporie.notebook.tabs.notebook.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.notebook.tabs.notebook.**Notebook**(*app: BaseApp*, *path: Path | None = None*, *kernel: Kernel | None = None*, *comms: dict[str, Comm] | None = None*, *use_kernel_history: bool = True*, *json: dict[str, Any] | None = None*)

Bases: *BaseNotebook*

Interactive notebooks.

A tab which allows running and editing a notebook.

```
NOTEBOOK_EXTENSIONS = ['.ipynb', '.md', '.markdown', '.Rmd', '.py',
'.coco', '.R', '.r', '.jl', '.cpp', '.ss', '.clj', '.scm', '.sh', '.ps1',
'.q', '.m', '.wolfram', '.pro', '.js', '.ts', '.scala', '.rs', '.robot',
'.resource', '.cs', '.fsx', '.fs', '.sos', '.java', '.groovy', '.sage',
'.ml', '.hs', '.tcl', '.mac', '.gp', '.do', '.sas', '.xsh', '.qmd',
'.myst', '.mystnb', '.mnb']
```

**add**(*index: int*, *source: str = ''*, ***kwargs: Any*) → None

Create a new cell at a given index.

**Parameters**

- **index** – The position at which to insert a new cell
- **source** – The contents of the new cell
- **kwargs** – Additional parameters for the cell

**add_cell_above**() → None

Inert a cell above the current selection.

**add_cell_below**() → None

Inert a cell below the current selection.

**allow_stdin: bool = True**

**bg_init = True**

**property cell: *Cell***

Return the currently selected *Cell* in this *Notebook*.

**property cells: Sequence[*Cell*]**

Return the currently selected *Cells* in this *Notebook*.

**change_kernel**(*msg: str | None = None*, *startup: bool = False*) → None

Prompt the user to select a new kernel.

**check_edit_mode**() → bool

Determine if the notebook is (or should be) in edit mode.

**clipboard: list[*Cell*]**

**close**(*cb: Callable | None = None*) → None

Check if the user want to save an unsaved notebook, then close the file.

**Parameters**

**cb** – A callback to run if after closing the notebook.

**comm_close**(*content: dict*, *buffers: Sequence[bytes]*) → None

Close a notebook Comm.

**comm_msg**(*content: dict*, *buffers: Sequence[bytes]*) → None

Respond to a Comm message from the kernel.

**comm_open**(*content: dict*, *buffers: Sequence[bytes]*) → None

Register a new kernel Comm object in the notebook.

**comms: dict[str, *Comm*]**

**completers: list[*Completer*]**

**container: AnyContainer**

**copy**(*slice_: slice | None = None*) → None

Add a copy of the selected cells to the *Notebook*'s clipboard.

**copy_outputs**(*slice_: slice | None = None*) → None

Copy the outputs of the selected cells.

**property current_input: *KernelInput***

Return the currently active kernel input, if any.

**cut**(*slice_: slice | None = None*) → None

Remove cells from the notebook and them to the *Notebook*'s clipboard.

**default_callbacks: *MsgCallbacks***

**delete**(*slice_: slice | None = None*) → None

Delete cells from the notebook.

**edit_mode = False**

**enter_edit_mode**() → None

Enter cell edit mode.

**exit_edit_mode**() → None

Leave cell edit mode.

**file_extensions: ClassVar[dict[str, None]] = {'.R': None, '.Rmd': None, '.clj': None, '.coco': None, '.cpp': None, '.cs': None, '.do': None, '.fs': None, '.fsx': None, '.gp': None, '.groovy': None, '.hs': None, '.ipynb': None, '.java': None, '.jl': None, '.js': None, '.m': None, '.mac': None, '.markdown': None, '.md': None, '.ml': None, '.mnb': None, '.myst': None, '.mystnb': None, '.pro': None, '.ps1': None, '.py': None, '.q': None, '.qmd': None, '.r': None, '.resource': None, '.robot': None, '.rs': None, '.sage': None, '.sas': None, '.scala': None, '.scm': None, '.sh': None, '.sos': None, '.ss': None, '.tcl': None, '.ts': None, '.wolfram': None, '.xsh': None}**

**focus**() → None

Focus the tab (or make it visible).

**formatters: list[*Formatter*]**

**get_cell_by_id**(*cell_id: str*) → *euporie.core.widgets.cell.Cell* | None

Return a reference to the *Cell* container with a given cell id.

**history: *History***

**init_kernel**(*kernel:* Kernel | *None = None*, *comms: dict[str,* Comm*] | None = None*, *use_kernel_history: bool = False*, *connection_file:* Path | *None = None*) → None

Set up the tab's kernel and related components.

**inspectors: list[*Inspector*]**

**interrupt_kernel**() → None

Interrupt the current *Notebook*'s kernel.

**kernel: *Kernel***

**kernel_died**() → None
Call if the kernel dies.

**property kernel_display_name: str**
Return the display name of the kernel defined in the notebook JSON.

**property kernel_lang_file_ext: str**
Return the display name of the kernel defined in the notebook JSON.

**kernel_language: str**

**property kernel_name: str**
Return the name of the kernel defined in the notebook JSON.

**kernel_started**(*result: dict[str, Any] | None = None*) → None
Run when the kernel has started.

**lang_file_ext**() → str
Return the file extension for scripts in the notebook's language.

**property language: str**
Return the name of the kernel defined in the notebook JSON.

**load**() → None
Load the notebook file from the file-system.

**load_container**() → AnyContainer
Load the main notebook container.

**async load_history**() → None
Load kernel history.

**async load_lsps**() → None
Load the LSP clients.

**load_widgets_from_metadata**() → None
Load widgets from state saved in notebook metadata.

**lsp_after_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we saved a document.

**lsp_before_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we are about to save a document.

**lsp_change_handler**(*lsp:* LspClient) → None
Tell the LSP server a file metadata has changed.

**lsp_close_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_open_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_update_diagnostics**(*lsp:* LspClient) → None
Process a new diagnostic report from the LSP.

**lsps: list[*LspClient*]**

**merge**(*slice_: slice | None = None*) → None

Merge two or more cells.

**property metadata: dict[str, Any]**

Return a dictionary to hold notebook / kernel metadata.

**mime_types: ClassVar[set[str]] = {'application/x-ipynb+json'}**

**mode**() → str

Return a symbol representing the current mode.

- ^: Notebook mode
- >: Navigation mode
- I: Insert mode
- v: Visual mode

**Returns**

A character representing the current mode

**move**(*n: int*, *slice_: slice | None = 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

**multiple_cells_selected: *Filter***

**name: str | None = 'Notebook Editor'**

**paste**(*index: int | None = None*) → None

Append the contents of the *Notebook*'s clipboard below the current cell.

**property path_name: str**

Return the path name.

**post_init_kernel**() → None

Start the kernel after if has been loaded.

**pre_init_kernel**() → None

Run stuff before the kernel is loaded.

**reformat**() → None

Reformat all code cells in the notebooks.

**refresh**(*slice_: slice | None = None*, *scroll: bool = True*) → None

Refresh the rendered contents of this notebook.

**refresh_cell**(*cell:* Cell) → None

Trigger the refresh of a notebook cell.

**rendered_cells**() → list[*euporie.core.widgets.cell.Cell*]

Return a list of rendered notebooks' cells.

**report**() → *Report*

Return the current diagnostic reports.

**report_kernel_error**(*error: Exception | None*) → None

Report a kernel error to the user.

**reports**: ***WeakKeyDictionary*[*LspClient*, *Report*]**

**reset**() → None

Reload the notebook file from the disk and re-render.

**restart_kernel**(*cb: Callable | None = None*) → None

Restart the current *Notebook*'s kernel.

**run_all**(*wait: bool = False*) → None

Run all cells.

**run_cell**(*cell:* Cell, *wait: bool = False*, *callback: Callable[..., None] | None = None*) → None

Run a cell.

**Parameters**

- **cell** – The rendered cell to run. If `None`, runs the currently selected cell.
- **wait** – If `True`, blocks until cell execution is finished
- **callback** – Function to run after completion

**run_selected_cells**(*advance: bool = False*, *insert: bool = False*) → None

Run the currently selected cells.

**Parameters**

- **advance** – If True, move to next cell. If True and at the last cell, create a new cell at the end of the notebook.
- **insert** – If True, add a new empty cell below the current cell and select it.

**save**(*path:* Path *| None = None*, *cb: Callable | None = None*) → None

Write the notebook's JSON to the current notebook's file.

Additionally save the widget state to the notebook metadata.

**Parameters**

- **path** – An optional new path at which to save the tab
- **cb** – A callback to run if after saving the notebook.

**scroll_to**(*index: int*) → None

Scroll to a cell by index.

**select**(*index: int*, *extend: bool = False*, *position: int | None = None*, *scroll: bool = True*) → None

Select a cell or adds it to the selection.

**Parameters**

- **index** – The index of the cell to select
- **extend** – If true, the selection will be extended to include the cell
- **position** – An optional cursor position index to apply to the cell input
- **scroll** – Whether to scroll the page

**property selected_indices: list[int]**

Return a list of the currently selected cell indices.

**set_kernel_info**(*info: dict*) → None

Handle kernel info requests.

**set_next_input**(*text: str*, *replace: bool = False*) → None

Handle `set_next_input` payloads, e.g. `%load` magic.

**set_status**(*status: str*) → None

Call when kernel status changes.

**split_cell**() → None

Split a cell into two at the cursor position.

**suggester: *AutoSuggest***

**property title: str**

Return the tab title.

**undelete**() → None

Insert the last deleted cell(s) back into the notebook.

**undo_buffer: *deque*[tuple[int, list[*Cell*]]]**

**weight: int = 3**

## Classes

| | |
|---|---|
| *DisplayTab*(app[, path]) | Tab class for displaying files. |
| *EditorTab*(app[, path, kernel, comms, ...]) | Tab class for editing text files. |
| *JsonTab*(app[, path]) | Tab class for JSON data. |
| *LogView*(app[, path]) | A tab which allows you to view log entries. |
| *Notebook*(app[, path, kernel, comms, ...]) | Interactive notebooks. |
| *WebTab*(app, path) | Tab class for displaying files. |

### euporie.notebook.tabs.DisplayTab

**class** euporie.notebook.tabs.**DisplayTab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Tab class for displaying files.

### euporie.notebook.tabs.EditorTab

**class** euporie.notebook.tabs.**EditorTab**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool = False*)

Tab class for editing text files.

### euporie.notebook.tabs.JsonTab

**class** euporie.notebook.tabs.**JsonTab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Tab class for JSON data.

### euporie.notebook.tabs.LogView

**class** euporie.notebook.tabs.**LogView**(*app:* BaseApp, *path:* Path | *None* = *None*)

A tab which allows you to view log entries.

### euporie.notebook.tabs.Notebook

**class** euporie.notebook.tabs.**Notebook**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *True*, *json: dict[str, Any]* | *None* = *None*)

Interactive notebooks.

A tab which allows running and editing a notebook.

### euporie.notebook.tabs.WebTab

**class** euporie.notebook.tabs.**WebTab**(*app:* BaseApp, *path:* Path | *None*)

Tab class for displaying files.

**class** euporie.notebook.tabs.**DisplayTab**(*app:* BaseApp, *path:* Path | *None* = *None*)

Bases: *Tab*

Tab class for displaying files.

**close**(*cb: Callable* | *None* = *None*) → None

Close a tab with a callback.

**Parameters**

**cb** – A function to call after the tab is closed.

**container: AnyContainer**

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None

Focus the tab (or make it visible).

**load_container**() → AnyContainer

Abcract method for loading the notebook's main container.

```
mime_types: ClassVar[set[str]] = {'*', 'application/pdf',
'application/x-latex', 'image/gif', 'image/jpeg', 'image/png',
'image/svg+xml', 'stream/std*', 'text/*', 'text/html', 'text/latex',
'text/markdown', 'text/x-markdown'}
```

**name: str | None = 'File Viewer'**

**reset**() → None

Reset the state of the tab.

**save**(*path:* Path | *None* = *None*, *cb: Callable* | *None* = *None*) → None

Save the current notebook.

**property title: str**

Return the tab title.

**weight: int = 0**

**class** euporie.notebook.tabs.**EditorTab**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *False*)

Bases: *KernelTab*

Tab class for editing text files.

**allow_stdin: bool = True**

**bg_init = True**

**change_kernel**(*msg: str* | *None* = *None*, *startup: bool* = *False*) → None

Prompt the user to select a new kernel.

**close**(*cb: Callable* | *None* = *None*) → None

Check if the user want to save an unsaved notebook, then close the file.

**Parameters**

**cb** – A callback to run if after closing the notebook.

**comm_close**(*content: dict*, *buffers: Sequence[bytes]*) → None

Close a notebook Comm.

**comm_msg**(*content: dict*, *buffers: Sequence[bytes]*) → None

Respond to a Comm message from the kernel.

**comm_open**(*content: dict*, *buffers: Sequence[bytes]*) → None

Register a new kernel Comm object in the notebook.

**comms: dict[str, *Comm*]**

**completers: list[*Completer*]**

**container: AnyContainer**

**property current_input: *KernelInput***

Return the currently active kernel input, if any.

**default_callbacks: *MsgCallbacks***

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None

Focus the tab (or make it visible).

**formatters: list[*Formatter*]**

**history: *History***

**init_kernel**(*kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history:* *bool* = *False*, *connection_file:* Path | *None* = *None*) → None

Set up the tab's kernel and related components.

**inspectors: list[*Inspector*]**

**interrupt_kernel**() → None

Interrupt the current *Notebook*'s kernel.

**kernel: *Kernel***

**property kernel_display_name: str**

Return the display name of the kernel defined in the notebook JSON.

**property kernel_lang_file_ext: str**

Return the display name of the kernel defined in the notebook JSON.

**kernel_language: str**

**property kernel_name: str**

Return the name of the kernel defined in the notebook JSON.

**kernel_started**(*result: dict[str, Any]* | *None* = *None*) → None

Task to run when the kernel has started.

**property language: str**

Return the name of the kernel defined in the notebook JSON.

**load**() → None

Load the text file.

**load_container**() → AnyContainer

Load the "tab"'s main container.

**async load_history**() → None

Load kernel history.

**async load_lsps**() → None

Load the LSP clients.

**lsp_after_save_handler**(*lsp:* LspClient) → None

Tell the the LSP we saved a document.

**lsp_before_save_handler**(*lsp:* LspClient) → None

Tell the the LSP we are about to save a document.

**lsp_change_handler**(*lsp:* LspClient) → None

Tell the LSP server a file has changed.

**lsp_close_handler**(*lsp:* LspClient) → None

Tell the LSP we opened a file.

**lsp_open_handler**(*lsp:* LspClient) → None

Tell the LSP we opened a file.

**lsp_update_diagnostics**(*lsp:* LspClient) → None

Process a new diagnostic report from the LSP.

**lsps**: list[*LspClient*]

**property metadata**: dict[str, Any]

Return a dictionary to hold notebook / kernel metadata.

**mime_types**: ClassVar[set[str]] = {'text/*'}

**name**: str | None = 'Text Editor'

**property path_name**: str

Return the path name.

**property position**: str

Return the position of the cursor in the document.

**post_init_kernel**() → None

Load UI and file in background after kernel has inited.

**pre_init_kernel**() → None

Run stuff before the kernel is loaded.

**report**() → *Report*

Return the current diagnostic reports.

**report_kernel_error**(*error: Exception | None*) → None

Report a kernel error to the user.

**reports**: *WeakKeyDictionary*[*LspClient*, *Report*]

**reset**() → None

Reset the state of the tab.

**restart_kernel**(*cb: Callable | None = None*) → None

Restart the current *Notebook*'s kernel.

**save**(*path:* Path | *None = None*, *cb: Callable | None = None*) → None

Save the current file.

**set_kernel_info**(*info: dict*) → None

Handle kernel info requests.

**suggester**: *AutoSuggest*

**property title**: str

Return the tab title.

**weight**: int = 1

**class** euporie.notebook.tabs.**JsonTab**(*app:* BaseApp, *path:* Path | *None = None*)

Bases: *Tab*

Tab class for JSON data.

**close**(*cb: Callable | None = None*) → None

Close a tab with a callback.

**Parameters**

**cb** – A function to call after the tab is closed.

**container: AnyContainer**

**file_extensions: ClassVar[dict[str, None]] = {}**

**filte_types: ClassVar[set[str]] = {'.json'}**

**focus**() → None

Focus the tab (or make it visible).

**load_container**() → AnyContainer

Abcract method for loading the notebook's main container.

**mime_types: ClassVar[set[str]] = {'*json'}**

**name: str | None = 'JSON Viewer'**

**reset**() → None

Reset the state of the tab.

**save**(*path:* Path | *None* = *None*, *cb: Callable* | *None* = *None*) → None

Save the current notebook.

**property title: str**

Return the tab title.

**weight: int = 0**

**class** euporie.notebook.tabs.**LogView**(*app:* BaseApp, *path:* Path | *None* = *None*)

Bases: *Tab*

A tab which allows you to view log entries.

**add_record**(*message:* FormattedText) → None

Add a single new record to the textarea.

**Parameters**

**message** – The formatted log record to add

**close**(*cb: Callable* | *None* = *None*) → None

Remove log queue handler hook on close.

**container: AnyContainer**

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None

Focus the tab (or make it visible).

**mime_types: ClassVar[set[str]] = {}**

**name: str | None = None**

**reset**() → None

Reset the state of the tab.

**save**(*path:* Path | *None* = *None*, *cb: Callable* | *None* = *None*) → None

Save the current notebook.

**property title: str**

Return the title of this tab.

**weight: int = 0**

**class** euporie.notebook.tabs.**Notebook**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *True*, *json: dict[str, Any]* | *None* = *None*)

Bases: *BaseNotebook*

Interactive notebooks.

A tab which allows running and editing a notebook.

**NOTEBOOK_EXTENSIONS = ['.ipynb', '.md', '.markdown', '.Rmd', '.py', '.coco', '.R', '.r', '.jl', '.cpp', '.ss', '.clj', '.scm', '.sh', '.ps1', '.q', '.m', '.wolfram', '.pro', '.js', '.ts', '.scala', '.rs', '.robot', '.resource', '.cs', '.fsx', '.fs', '.sos', '.java', '.groovy', '.sage', '.ml', '.hs', '.tcl', '.mac', '.gp', '.do', '.sas', '.xsh', '.qmd', '.myst', '.mystnb', '.mnb']**

**add**(*index: int*, *source: str* = '', ***kwargs: Any*) → None

Create a new cell at a given index.

**Parameters**

- **index** – The position at which to insert a new cell
- **source** – The contents of the new cell
- **kwargs** – Additional parameters for the cell

**add_cell_above**() → None

Inert a cell above the current selection.

**add_cell_below**() → None

Inert a cell below the current selection.

**allow_stdin: bool = True**

**bg_init = True**

**property cell: *Cell***

Return the currently selected *Cell* in this *Notebook*.

**property cells: Sequence[*Cell*]**

Return the currently selected *Cells* in this *Notebook*.

**change_kernel**(*msg: str* | *None* = *None*, *startup: bool* = *False*) → None

Prompt the user to select a new kernel.

**check_edit_mode**() → bool

Determine if the notebook is (or should be) in edit mode.

**close**(*cb: Callable* | *None* = *None*) → None

Check if the user want to save an unsaved notebook, then close the file.

**Parameters**

**cb** – A callback to run if after closing the notebook.

**comm_close**(*content: dict*, *buffers: Sequence[bytes]*) → None

Close a notebook Comm.

**comm_msg**(*content: dict*, *buffers: Sequence[bytes]*) → None

Respond to a Comm message from the kernel.

**comm_open**(*content: dict*, *buffers: Sequence[bytes]*) → None

Register a new kernel Comm object in the notebook.

**comms: dict[str, *Comm*]**

**completers: list[*Completer*]**

**container: AnyContainer**

**copy**(*slice_: slice | None = None*) → None

Add a copy of the selected cells to the *Notebook*'s clipboard.

**copy_outputs**(*slice_: slice | None = None*) → None

Copy the outputs of the selected cells.

**property current_input: *KernelInput***

Return the currently active kernel input, if any.

**cut**(*slice_: slice | None = None*) → None

Remove cells from the notebook and them to the *Notebook*'s clipboard.

**default_callbacks: *MsgCallbacks***

**delete**(*slice_: slice | None = None*) → None

Delete cells from the notebook.

**edit_mode = False**

**enter_edit_mode**() → None

Enter cell edit mode.

**exit_edit_mode**() → None

Leave cell edit mode.

**file_extensions: ClassVar[dict[str, None]] = {'.R': None, '.Rmd': None, '.clj': None, '.coco': None, '.cpp': None, '.cs': None, '.do': None, '.fs': None, '.fsx': None, '.gp': None, '.groovy': None, '.hs': None, '.ipynb': None, '.java': None, '.jl': None, '.js': None, '.m': None, '.mac': None, '.markdown': None, '.md': None, '.ml': None, '.mnb': None, '.myst': None, '.mystnb': None, '.pro': None, '.ps1': None, '.py': None, '.q': None, '.qmd': None, '.r': None, '.resource': None, '.robot': None, '.rs': None, '.sage': None, '.sas': None, '.scala': None, '.scm': None, '.sh': None, '.sos': None, '.ss': None, '.tcl': None, '.ts': None, '.wolfram': None, '.xsh': None}**

**focus**() → None

Focus the tab (or make it visible).

**formatters: list[*Formatter*]**

**get_cell_by_id**(*cell_id: str*) → *euporie.core.widgets.cell.Cell* | None

Return a reference to the *Cell* container with a given cell id.

**history: *History***

**init_kernel**(*kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history:* *bool* = *False*, *connection_file:* Path | *None* = *None*) → None

Set up the tab's kernel and related components.

**inspectors: list[*Inspector*]**

**interrupt_kernel**() → None

Interrupt the current *Notebook*'s kernel.

**kernel: *Kernel***

**kernel_died**() → None

Call if the kernel dies.

**property kernel_display_name: str**

Return the display name of the kernel defined in the notebook JSON.

**property kernel_lang_file_ext: str**

Return the display name of the kernel defined in the notebook JSON.

**kernel_language: str**

**property kernel_name: str**

Return the name of the kernel defined in the notebook JSON.

**kernel_started**(*result: dict[str, Any]* | *None* = *None*) → None

Run when the kernel has started.

**lang_file_ext**() → str

Return the file extension for scripts in the notebook's language.

**property language: str**

Return the name of the kernel defined in the notebook JSON.

**load**() → None

Load the notebook file from the file-system.

**load_container**() → AnyContainer

Load the main notebook container.

**async load_history**() → None

Load kernel history.

**async load_lsps**() → None

Load the LSP clients.

**load_widgets_from_metadata**() → None

Load widgets from state saved in notebook metadata.

**lsp_after_save_handler**(*lsp:* LspClient) → None

Tell the the LSP we saved a document.

**lsp_before_save_handler**(*lsp:* LspClient) → None

Tell the the LSP we are about to save a document.

**lsp_change_handler**(*lsp:* LspClient) → None

Tell the LSP server a file metadata has changed.

**lsp_close_handler**(*lsp:* LspClient) → None

Tell the LSP we opened a file.

**lsp_open_handler**(*lsp:* LspClient) → None

Tell the LSP we opened a file.

**lsp_update_diagnostics**(*lsp:* LspClient) → None

Process a new diagnostic report from the LSP.

**lsps: list[*LspClient*]**

**merge**(*slice_: slice | None = None*) → None

Merge two or more cells.

**property metadata: dict[str, Any]**

Return a dictionary to hold notebook / kernel metadata.

**mime_types: ClassVar[set[str]] = {'application/x-ipynb+json'}**

**mode**() → str

Return a symbol representing the current mode.

- ^: Notebook mode
- >: Navigation mode
- I: Insert mode
- v: Visual mode

**Returns**

A character representing the current mode

**move**(*n: int, slice_: slice | None = 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

**multiple_cells_selected: *Filter***

**name: str | None = 'Notebook Editor'**

**paste**(*index: int | None = None*) → None

Append the contents of the *Notebook*'s clipboard below the current cell.

**property path_name: str**

Return the path name.

**post_init_kernel**() → None

Start the kernel after if has been loaded.

**pre_init_kernel**() → None

Run stuff before the kernel is loaded.

**reformat**() → None

Reformat all code cells in the notebooks.

**refresh**(*slice_: slice | None = None*, *scroll: bool = True*) → None

Refresh the rendered contents of this notebook.

**refresh_cell**(*cell:* Cell) → None

Trigger the refresh of a notebook cell.

**rendered_cells**() → list[*euporie.core.widgets.cell.Cell*]

Return a list of rendered notebooks' cells.

**report**() → *Report*

Return the current diagnostic reports.

**report_kernel_error**(*error: Exception | None*) → None

Report a kernel error to the user.

**reports**: *WeakKeyDictionary*[*LspClient*, *Report*]

**reset**() → None

Reload the notebook file from the disk and re-render.

**restart_kernel**(*cb: Callable | None = None*) → None

Restart the current *Notebook*'s kernel.

**run_all**(*wait: bool = False*) → None

Run all cells.

**run_cell**(*cell:* Cell, *wait: bool = False*, *callback: Callable[..., None] | None = None*) → None

Run a cell.

**Parameters**

- **cell** – The rendered cell to run. If `None`, runs the currently selected cell.
- **wait** – If `True`, blocks until cell execution is finished
- **callback** – Function to run after completion

**run_selected_cells**(*advance: bool = False*, *insert: bool = False*) → None

Run the currently selected cells.

**Parameters**

- **advance** – If True, move to next cell. If True and at the last cell, create a new cell at the end of the notebook.
- **insert** – If True, add a new empty cell below the current cell and select it.

**save**(*path:* Path *| None = None*, *cb: Callable | None = None*) → None

Write the notebook's JSON to the current notebook's file.

Additionally save the widget state to the notebook metadata.

**Parameters**

- **path** – An optional new path at which to save the tab
- **cb** – A callback to run if after saving the notebook.

**scroll_to**(*index: int*) → None

Scroll to a cell by index.

**select**(*index: int*, *extend: bool* = *False*, *position: int* | *None* = *None*, *scroll: bool* = *True*) → None

Select a cell or adds it to the selection.

**Parameters**

- **index** – The index of the cell to select
- **extend** – If true, the selection will be extended to include the cell
- **position** – An optional cursor position index to apply to the cell input
- **scroll** – Whether to scroll the page

**property selected_indices: list[int]**

Return a list of the currently selected cell indices.

**set_kernel_info**(*info: dict*) → None

Handle kernel info requests.

**set_next_input**(*text: str*, *replace: bool* = *False*) → None

Handle `set_next_input` payloads, e.g. `%load` magic.

**set_status**(*status: str*) → None

Call when kernel status changes.

**split_cell**() → None

Split a cell into two at the cursor position.

**suggester: *AutoSuggest***

**property title: str**

Return the tab title.

**undelete**() → None

Insert the last deleted cell(s) back into the notebook.

**weight: int = 3**

**class** euporie.notebook.tabs.**WebTab**(*app:* BaseApp, *path:* Path | *None*)

Bases: *Tab*

Tab class for displaying files.

**close**(*cb: Callable* | *None* = *None*) → None

Close a tab with a callback.

**Parameters**

**cb** – A function to call after the tab is closed.

**container: AnyContainer**

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None

Focus the webview when this tab is focused.

**load_container**() → AnyContainer

Abcract method for loading the notebook's main container.

**load_url**(*url: str* | Path, *new_tab: bool* = *False*, ***kwargs: Any*) → bool

Load a new URL, or the URL in the address-bar.

**mime_types: ClassVar[set[str]] = {'text/html', 'text/markdown'}**

**name: str | None = 'Web Viewer'**

**reset**() → None
Reset the state of the tab.

**save**(*path:* Path | *None* = *None*, *cb: Callable* | *None* = *None*) → None
Save the current notebook.

**property title: str**
Return the tab title.

**weight: int = 2**

### euporie.notebook.widgets

Contain widgets used in the notebook app.

### Modules

| | |
|---|---|
| *euporie.notebook.widgets.side_bar* | Define a side-bar. |

### euporie.notebook.widgets.side_bar

Define a side-bar.

### Functions

| | |
|---|---|
| *add_cmd*(**kwargs) | Add a command to the centralized command system. |
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *register_bindings*(bindings) | Update the key-binding registry. |

### euporie.notebook.widgets.side_bar.add_cmd

euporie.notebook.widgets.side_bar.**add_cmd**(***kwargs: Any*) → Callable
Add a command to the centralized command system.

### euporie.notebook.widgets.side_bar.add_setting

euporie.notebook.widgets.side_bar.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any]* | *None* = *None*, *action: argparse.Action* | *str* | *None* = *None*, *flags: list[str]* | *None* = *None*, *schema: dict[str, Any]* | *None* = *None*, *nargs: str* | *int* | *None* = *None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[*Setting*], None]]* | *None* = *None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.notebook.widgets.side_bar.register_bindings

euporie.notebook.widgets.side_bar.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### Classes

| | |
|---|---|
| *Condition*(func) | Turn any callable into a Filter. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *DynamicContainer*(get_container) | Container class that dynamically returns any Container. |
| *FormattedTextControl*([text, style, ...]) | Control that displays formatted text. |
| *HSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked above/under the other. |
| *Line*([char, width, height, collapse, style]) | Draw a horizontal or vertical line. |
| *SideBar*(titles, icons, panels) | A side-bar for the notebook application. |
| *SideBarButtons*(options, labels, index, ...) | Vertical toggle-buttons with additional styling for the side-bar. |
| *ToggleButton*(text, on_click, ...) | A toggleable button widget. |
| *ToggleButtons*(options, labels, index, ...) | A widget where an option is selected using mutually exclusive toggle-buttons. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *WindowAlign*(value[, names, module, ...]) | Alignment of the Window content. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.notebook.widgets.side_bar.Condition

**class** euporie.notebook.widgets.side_bar.**Condition**(*func: Callable[[], bool]*)

Turn any callable into a Filter. The callable is supposed to not take any arguments.

This can be used as a decorator:

```
@Condition
def feature_is_active():  # `feature_is_active` becomes a Filter.
    return True
```

**Parameters**

**func** – Callable which takes no inputs and returns a boolean.

### euporie.notebook.widgets.side_bar.ConditionalContainer

**class** euporie.notebook.widgets.side_bar.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.notebook.widgets.side_bar.DynamicContainer

**class** euporie.notebook.widgets.side_bar.**DynamicContainer**(*get_container: Callable[[], AnyContainer]*)

Container class that dynamically returns any Container.

**Parameters**

**get_container** – Callable that returns a *Container* instance or any widget with a `__pt_container__` method.

### euporie.notebook.widgets.side_bar.FormattedTextControl

**class** euporie.notebook.widgets.side_bar.**FormattedTextControl**(*text: AnyFormattedText = ''*, *style: str = ''*, *focusable: FilterOrBool = False*, *key_bindings: KeyBindingsBase | None = None*, *show_cursor: bool = True*, *modal: bool = False*, *get_cursor_position: Callable[[], Point | None] | None = None*)

Control that displays formatted text. This can be either plain text, an `HTML` object an `ANSI` object, a list of `(style_str, text)` tuples or a callable that takes no argument and returns one of those, depending on how you prefer to do the formatting. See `prompt_toolkit.layout.formatted_text` for more information.

(It's mostly optimized for rather small widgets, like toolbars, menus, etc…)

When this UI control has the focus, the cursor will be shown in the upper left corner of this control by default. There are two ways for specifying the cursor position:

- Pass a *get_cursor_position* function which returns a *Point* instance with the current cursor position.
- If the (formatted) text is passed as a list of `(style, text)` tuples and there is one that looks like `('[SetCursorPosition]', '')`, then this will specify the cursor position.

Mouse support:

The list of fragments can also contain tuples of three items, looking like: (style_str, text, handler). When mouse support is enabled and the user clicks on this fragment, then the given handler is called. That handler should accept two inputs: (Application, MouseEvent) and it should either handle the event or return *NotImplemented* in case we want the containing Window to handle this event.

**Parameters**

- **focusable** – *bool* or *Filter*: Tell whether this control is focusable.
- **text** – Text or formatted text to be displayed.
- **style** – Style string applied to the content. (If you want to style the whole Window, pass the style to the Window instead.)
- **key_bindings** – a *KeyBindings* object.
- **get_cursor_position** – A callable that returns the cursor position as a *Point* instance.

### euporie.notebook.widgets.side_bar.HSplit

**class** euporie.notebook.widgets.side_bar.**HSplit**(*children: Sequence[AnyContainer]*, *window_too_small:* Container | *None* = *None*, *align: VerticalAlign* = *VerticalAlign.JUSTIFY*, *padding: AnyDimension* = *0*, *padding_char: str* | *None* = *None*, *padding_style: str* = *''*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *modal: bool* = *False*, *key_bindings:* KeyBindingsBase | *None* = *None*, *style: str* | *Callable[[], str]* = *''*)

Several layouts, one stacked above/under the other.

### euporie.notebook.widgets.side_bar.Line

**class** euporie.notebook.widgets.side_bar.**Line**(*char: str* | *None* = *None*, *width: int* | *None* = *None*, *height: int* | *None* = *None*, *collapse: bool* = *False*, *style: str* = *'class:grid-line'*)

Draw a horizontal or vertical line.

### euporie.notebook.widgets.side_bar.SideBar

**class** euporie.notebook.widgets.side_bar.**SideBar**(*titles: Sequence[str]*, *icons: Sequence[str]*, *panels: Sequence[AnyContainer]*)

A side-bar for the notebook application.

### euporie.notebook.widgets.side_bar.SideBarButtons

**class** euporie.notebook.widgets.side_bar.**SideBarButtons**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, disabled: FilterOrBool = False, vertical: FilterOrBool = False*)

Vertical toggle-buttons with additional styling for the side-bar.

### euporie.notebook.widgets.side_bar.ToggleButton

**class** euporie.notebook.widgets.side_bar.**ToggleButton**(*text: AnyFormattedText, on_click: Callable[[ToggleButton], None] | None = None, width: int | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, show_borders: DiBool | None = None, selected: bool = False, disabled: FilterOrBool = False, key_bindings: KeyBindingsBase | None = None*)

A toggleable button widget.

### euporie.notebook.widgets.side_bar.ToggleButtons

**class** euporie.notebook.widgets.side_bar.**ToggleButtons**(*options: list[Any], labels: Sequence[AnyFormattedText] | None = None, index: int | None = None, indices: list[int] | None = None, n_values: int | None = None, multiple: FilterOrBool = False, max_count: int | None = None, on_change: Callable[[SelectableWidget], None] | None = None, style: str | Callable[[], str] = 'class:input', border: GridStyle | None = □□ □ □□ □□□□ □□, disabled: FilterOrBool = False, vertical: FilterOrBool = False*)

A widget where an option is selected using mutually exclusive toggle-buttons.

### euporie.notebook.widgets.side_bar.VSplit

**class** euporie.notebook.widgets.side_bar.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.notebook.widgets.side_bar.Window

**class** euporie.notebook.widgets.side_bar.**Window**(*content: UIControl | None = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *dont_extend_width: FilterOrBool = False*, *dont_extend_height: FilterOrBool = False*, *ignore_content_width: FilterOrBool = False*, *ignore_content_height: FilterOrBool = False*, *left_margins: Sequence[Margin] | None = None*, *right_margins: Sequence[Margin] | None = None*, *scroll_offsets: ScrollOffsets | None = None*, *allow_scroll_beyond_bottom: FilterOrBool = False*, *wrap_lines: FilterOrBool = False*, *get_vertical_scroll: Callable[[Window], int] | None = None*, *get_horizontal_scroll: Callable[[Window], int] | None = None*, *always_hide_cursor: FilterOrBool = False*, *cursorline: FilterOrBool = False*, *cursorcolumn: FilterOrBool = False*, *colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None*, *align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT*, *style: str | Callable[[], str] = ''*, *char: None | str | Callable[[], str] = None*, *get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

### euporie.notebook.widgets.side_bar.WindowAlign

**class** euporie.notebook.widgets.side_bar.**WindowAlign**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Alignment of the Window content.

Note that this is different from *HorizontalAlign* and *VerticalAlign*, which are used for the alignment of the child containers in respectively *VSplit* and *HSplit*.

### euporie.notebook.widgets.side_bar.partial

**class** euporie.notebook.widgets.side_bar.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.notebook.widgets.side_bar.**SideBar**(*titles: Sequence[str]*, *icons: Sequence[str]*, *panels: Sequence[AnyContainer]*)

Bases: object

A side-bar for the notebook application.

**toggle_pane**() → None

Toggle the visibility of the side-bar.

**class** euporie.notebook.widgets.side_bar.**SideBarButtons**(*options: list[Any]*, *labels: Sequence[AnyFormattedText] | None = None*, *index: int | None = None*, *indices: list[int] | None = None*, *n_values: int | None = None*, *multiple: FilterOrBool = False*, *max_count: int | None = None*, *on_change: Callable[[SelectableWidget], None] | None = None*, *style: str | Callable[[], str] = 'class:input'*, *border: GridStyle | None = □□ □□□ □□□□□ □□*, *disabled: FilterOrBool = False*, *vertical: FilterOrBool = False*)

Bases: *ToggleButtons*

Vertical toggle-buttons with additional styling for the side-bar.

**get_button_style**(*index: int*) → Callable[[], str]

Return the current button style.

**hover_rel**(*rel: int*) → None

Hover an index relative to the current hovered index.

**property index: int | None**

Return the first selected index.

**property indices: list[int]**

Return a list of the selected indices.

**key_bindings**() → *KeyBindingsBase*

Return key-bindings for the drop-down widget.

**load_container**() → AnyContainer

Load the widget's container.

**property mask: list[bool]**

Get mask of selected options.

**mouse_handler**(*i: int*, *mouse_event:* MouseEvent) → NotImplementedOrNone

Handle mouse events.

**select_rel**(*rel: int*) → None

Select an index relative to the current index.

**property style: str**

Return the widget's style.

**toggle_item**(*index: int*) → None

Toggle the selection status of the option at a given index.

**update_buttons**(*widget:* euporie.core.widgets.forms.SelectableWidget | *None* = *None*) → None

Set the toggle buttons' selection state when the selected index changes.

**property value: Any**

Return the selected value.

**property values: list[Any]**

Return a list of the selected values.

### 5.12.5 euporie.preview

A euporie app for previewing notebook files.

#### Modules

| | |
|---|---|
| *euporie.preview.app* | Concern dumping output. |
| *euporie.preview.tabs* | Notebook tab for use in preview app. |

#### euporie.preview.app

Concern dumping output.

## Functions

| | |
|---|---|
| *add_setting*(name, default, help_, description) | Register a new config item. |
| *cast*(typ, val) | Cast a value to a type. |
| *create_output*([stdout, always_prefer_tty]) | Return an Output instance for the command line. |
| *get_app*() | Get the current active (running) Application. |
| *get_preview_app*() | Get the current application. |
| *register_bindings*(bindings) | Update the key-binding registry. |

### euporie.preview.app.add_setting

euporie.preview.app.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### euporie.preview.app.cast

euporie.preview.app.**cast**(*typ*, *val*)

Cast a value to a type.

This returns the value unchanged. To the type checker this signals that the return value has the designated type, but at runtime we intentionally don't check anything (we want this to be as fast as possible).

### euporie.preview.app.create_output

euporie.preview.app.**create_output**(*stdout: TextIO | None = None*, *always_prefer_tty: bool = False*) → Output

Return an Output instance for the command line.

**Parameters**

- **stdout** – The stdout object
- **always_prefer_tty** – When set, look for *sys.stderr* if *sys.stdout* is not a TTY. Useful if *sys.stdout* is redirected to a file, but we still want user input and output on the terminal.

  By default, this is *False*. If *sys.stdout* is not a terminal (maybe it's redirected to a file), then a *PlainTextOutput* will be returned. That way, tools like *print_formatted_text* will write plain text into that file.

### euporie.preview.app.get_app

euporie.preview.app.**get_app**() → *BaseApp*

Get the current active (running) Application.

### euporie.preview.app.get_preview_app

euporie.preview.app.**get_preview_app**() → *PreviewApp*

Get the current application.

### euporie.preview.app.register_bindings

euporie.preview.app.**register_bindings**(*bindings: dict[str, KeyBindingDefs]*) → None

Update the key-binding registry.

### Classes

| | |
|---|---|
| *BaseApp*([title, set_title, leave_graphics, ...]) | All euporie apps. |
| *DynamicContainer*(get_container) | Container class that dynamically returns any Container. |
| *FloatContainer*(content, floats[, modal, ...]) | A *FloatContainer* which uses :py`BoundedWritePosition`s. |
| *PreviewApp*(**kwargs) | Preview app. |
| *PreviewNotebook*(app[, path, use_kernel_history]) | A notebook tab which renders cells sequentially. |
| *PseudoTTY*(underlying[, isatty]) | Make an output stream look like a TTY. |
| *UPath*(*args[, protocol]) | |
| *Vt100_Output*(stdout, get_size[, term, ...]) | **param get_size**<br>A callable which returns the *Size* of the output terminal. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |
| *partial* | partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords. |

### euporie.preview.app.BaseApp

**class** euporie.preview.app.**BaseApp**(*title: str | None = None*, *set_title: bool = True*, *leave_graphics: FilterOrBool = True*, *extend_renderer_height: FilterOrBool = False*, *extend_renderer_width: FilterOrBool = False*, *enable_page_navigation_bindings: FilterOrBool | None = True*, ***kwargs: Any*)

All euporie apps.

The base euporie application class.

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

### euporie.preview.app.DynamicContainer

**class** euporie.preview.app.**DynamicContainer**(*get_container: Callable[[], AnyContainer]*)

Container class that dynamically returns any Container.

**Parameters**

**get_container** – Callable that returns a *Container* instance or any widget with a `__pt_container__` method.

### euporie.preview.app.FloatContainer

**class** euporie.preview.app.**FloatContainer**(*content: AnyContainer*, *floats: list[*Float*]*, *modal: bool = False*, *key_bindings:* KeyBindingsBase | *None = None*, *style: str | Callable[[], str] = ''*, *z_index: int | None = None*)

A *FloatContainer* which uses :py`BoundedWritePosition`s.

### euporie.preview.app.PreviewApp

**class** euporie.preview.app.**PreviewApp**(***kwargs: Any*)

Preview app.

Preview notebook files in the terminal.

Outputs a formatted notebook file. The formatted output will be written to the the output file path given by *output_file* (the standard output by default).

### euporie.preview.app.PreviewNotebook

**class** euporie.preview.app.**PreviewNotebook**(*app:* BaseApp, *path:* Path | *None = None*, *use_kernel_history: bool = False*)

A notebook tab which renders cells sequentially.

### euporie.preview.app.PseudoTTY

**class** euporie.preview.app.**PseudoTTY**(*underlying: IO[str]* | TextIO, *isatty: bool = True*)

Make an output stream look like a TTY.

### euporie.preview.app.UPath

**class** euporie.preview.app.**UPath**(**args*, *protocol: str | None = None*, ***storage_options: Any*)

### euporie.preview.app.Vt100_Output

**class** euporie.preview.app.**Vt100_Output**(*stdout:* TextIO, *get_size: Callable[[],* Size*]*, *term: str* | *None* = *None*, *default_color_depth:* prompt_toolkit.output.color_depth.ColorDepth | *None* = *None*, *enable_bell: bool* = *True*, *enable_cpr: bool* = *True*)

**Parameters**

- **get_size** – A callable which returns the *Size* of the output terminal.
- **stdout** – Any object with has a *write* and *flush* method + an 'encoding' property.
- **term** – The terminal environment variable. (xterm, xterm-256color, linux, …)
- **enable_cpr** – When *True* (the default), send "cursor position request" escape sequences to the output in order to detect the cursor position. That way, we can properly determine how much space there is available for the UI (especially for drop down menus) to render. The *Renderer* will still try to figure out whether the current terminal does respond to CPR escapes. When *False*, never attempt to send CPR requests.

### euporie.preview.app.Window

**class** euporie.preview.app.**Window**(*content:* UIControl | *None* = *None*, *width: AnyDimension* = *None*, *height: AnyDimension* = *None*, *z_index: int* | *None* = *None*, *dont_extend_width: FilterOrBool* = *False*, *dont_extend_height: FilterOrBool* = *False*, *ignore_content_width: FilterOrBool* = *False*, *ignore_content_height: FilterOrBool* = *False*, *left_margins: Sequence[*Margin*]* | *None* = *None*, *right_margins: Sequence[*Margin*]* | *None* = *None*, *scroll_offsets:* ScrollOffsets | *None* = *None*, *allow_scroll_beyond_bottom: FilterOrBool* = *False*, *wrap_lines: FilterOrBool* = *False*, *get_vertical_scroll: Callable[[*Window*], int]* | *None* = *None*, *get_horizontal_scroll: Callable[[*Window*], int]* | *None* = *None*, *always_hide_cursor: FilterOrBool* = *False*, *cursorline: FilterOrBool* = *False*, *cursorcolumn: FilterOrBool* = *False*, *colorcolumns: None* | *list[ColorColumn]* | *Callable[[], list[ColorColumn]]* = *None*, *align:* WindowAlign | *Callable[[],* WindowAlign*]* = *WindowAlign.LEFT*, *style: str* | *Callable[[], str]* = *''*, *char: None* | *str* | *Callable[[], str]* = *None*, *get_line_prefix: GetLinePrefixCallable* | *None* = *None*)

Container that holds a control.

### euporie.preview.app.partial

**class** euporie.preview.app.**partial**

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

**class** euporie.preview.app.**PreviewApp**(***kwargs: Any*)

Bases: *BaseApp*

Preview app.

Preview notebook files in the terminal.

Outputs a formatted notebook file. The formatted output will be written to the the output file path given by *output_file* (the standard output by default).

**async cancel_and_wait_for_background_tasks**() → None

Cancel all background tasks, and wait for the cancellation to complete. 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.)

**cleanup**(*signum: int*, *frame: FrameType | None*) → None

Restore the state of the terminal on unexpected exit.

**cleanup_closed_tab**(*tab:* Tab) → None

Exit if all tabs are closed.

**close_tab**(*tab:* Tab | *None* = *None*) → None

Close a notebook tab.

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

**property 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.

**color_palette: *ColorPalette***

**context: contextvars.Context | None**

**cpr_not_supported_callback**() → None

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

**create_background_task**(*coroutine: Coroutine[Any, Any, None]*) → Task[None]

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

Given that we still support Python versions before 3.11, we can't use task groups (and exception groups), because of that, these background tasks are not allowed to raise exceptions. If they do, we'll call the default exception handler from the event loop.

If at some point, we have Python 3.11 as the minimum supported Python version, then we can use a *TaskGroup* (with the lifetime of *Application.run_async()*, and run run the background tasks in there.

This is not threadsafe.

**create_merged_style**() → *BaseStyle*

Generate a new merged style for the application.

Using a dynamic style has serious performance issues, so instead we update the style on the renderer directly when it changes in *self.update_style*

**Returns**
Return a combined style to use for the application

**property current_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: SearchState**

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

**dialogs: dict[str, *Dialog*]**

**draw**(*render_as_done: bool = True*) → None

Draw the app without focus, leaving the cursor below the drawn output.

**exit**(*result: _AppResult | None = None*, *exception: BaseException | type[BaseException] | None = None*, *style: str = ''*) → None

Optionally pipe the output to a pager on exit.

**focus_tab**(*tab:* Tab) → None

Make a tab visible and focuses it.

**focused_element: FocusableElement | None**

**formatters: list[*Formatter*]**

**full_screen: bool**

**future: Future[_AppResult] | None**

**get_edit_mode**() → *EditingMode*

Return the editing mode enum defined in the configuration.

**get_file_tab**(*path:* Path) → type[*Tab*]

Return the tab to use for a file path.

**get_file_tabs**(*path:* Path) → list[type[*Tab*]]

Return the tab to use for a file path.

**get_language_lsps**(*language: str*) → list[*euporie.core.lsp.LspClient*]

Return the approprrate LSP clients for a given language.

**get_used_style_strings**() → list[str]

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

**graphics: *WeakSet*[*Float*]**

**async classmethod interact**(*ssh_session:* PromptToolkitSSHSession) → None

Run the app asynchronously for the hub SSH server.

**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

Launch the app.

**load_container**() → *FloatContainer*

Return a container with all opened tabs.

**classmethod load_input**() → Input

Create the input for this application to use.

Ensures the TUI app always tries to run in a TTY.

**Returns**

A prompt-toolkit input instance

**load_key_bindings**() → None

Load the application's key bindings.

**classmethod load_output**() → Output

Load 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

**log_stdout_level: str = 'CRITICAL'**

**loop: AbstractEventLoop | None**

**lsp_clients: *WeakValueDictionary*[str, *LspClient*]**

**menus: dict[str, *Float*]**

**mouse_limits: *WritePosition* | None**

**mouse_position: *Point***

**name: str = 'preview'**

**open_file**(*path:* Path, *read_only: bool = False*, *tab_class: type[*Tab*] | None = None*) → None

Create 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
- **tab_class** – The tab type to use to open the file

**open_files**() → None

Open the files defined in the configuration.

**pager: *Pager* | None**

**pause_rendering**() → None

Block rendering, but allows input to be processed.

The first line prevents the display being drawn, and the second line means the key processor continues to process keys. We need this as we need to wait for the results of terminal queries which come in as key events.

This is used to prevent flicker when we update the styles based on terminal feedback.

**post_load**() → None

Allow subclasses to define additional loading steps.

**post_load_callables**: **list**[**Callable**[[], **None**]]

**pre_run**(*app:* prompt_toolkit.application.application.Application | *None* = *None*) → None

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

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

**print_text**(*text: AnyFormattedText*, *style:* BaseStyle | *None* = *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.

**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.

**resume_rendering**() → None

Reume rendering the app.

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

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

This will run the application in a fresh asyncio event loop.

**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_appp().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: Callable[[], None] | None = None*, *set_exception_handler: bool = True*, *handle_sigint: bool = True*, *slow_callback_duration: float = 0.5*) → _AppResult

Run the application.

**async run_system_command**(*command: str*, *wait_for_enter: bool = True*, *display_before_text: AnyFormattedText = ''*, *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.

**search_bar: *SearchBar* | None**

**shutdown_lsps**() → None

Shut down all the remaining LSP servers.

**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 syntax_theme: str**

Calculate the current syntax theme.

**property tab: *Tab* | None**

Return the currently selected tab container object.

**property tab_idx: int**

Get the current tab index.

**tabs: list[*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 uses 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.

**property title: str**

The application's title.

**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_edit_mode**(*setting:* Setting | *None* = *None*) → None

Set the keybindings for editing mode.

**update_style**(*query:* TerminalQuery | Setting | *None* = *None*) → None

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

**vi_state**

Vi state. (For Vi key bindings.)

euporie.preview.app.**get_preview_app**() → *PreviewApp*

Get the current application.

### euporie.preview.tabs

Notebook tab for use in preview app.

### Modules

| | |
|---|---|
| *euporie.preview.tabs.notebook* | A notebook which renders cells one cell at a time. |

### euporie.preview.tabs.notebook

A notebook which renders cells one cell at a time.

### Functions

| | |
|---|---|
| *add_setting*(name, default, help_, description) | Register a new config item. |

### euporie.preview.tabs.notebook.add_setting

euporie.preview.tabs.notebook.**add_setting**(*name: str*, *default: Any*, *help_: str*, *description: str*, *type_: Callable[[Any], Any] | None = None*, *action: argparse.Action | str | None = None*, *flags: list[str] | None = None*, *schema: dict[str, Any] | None = None*, *nargs: str | int | None = None*, *hidden: FilterOrBool = False*, *hooks: list[Callable[[Setting], None]] | None = None*, *cmd_filter: FilterOrBool = True*, ***kwargs: Any*) → None

Register a new config item.

### Classes

| | |
|---|---|
| *BaseNotebook*(app[, path, kernel, comms, ...]) | The main notebook container class. |
| *Box*(body[, padding, padding_left, ...]) | Add padding around a container. |
| *Cell*(index, json, kernel_tab[, is_new]) | A kernel_tab cell element. |
| *ConditionalContainer*(content, filter) | Wrapper around any other container that can change the visibility. |
| *Dimension*([min, max, weight, preferred]) | Specified dimension (width/height) of a user control or window. |
| *DynamicContainer*(get_container) | Container class that dynamically returns any Container. |
| *FastDictCache*(get_value[, size]) | Fast, lightweight cache which keeps at most *size* items. |
| *PreviewNotebook*(app[, path, use_kernel_history]) | A notebook tab which renders cells sequentially. |
| *PrintingContainer*(children[, width, ...]) | A container which displays all it's children in a vertical list. |
| *VSplit*(children[, window_too_small, align, ...]) | Several layouts, one stacked left/right of the other. |
| *Window*([content, width, height, z_index, ...]) | Container that holds a control. |

#### euporie.preview.tabs.notebook.BaseNotebook

**class** euporie.preview.tabs.notebook.**BaseNotebook**(*app:* BaseApp, *path:* Path | *None* = *None*, *kernel:* Kernel | *None* = *None*, *comms:* *dict[str,* Comm*]* | *None* = *None*, *use_kernel_history:* *bool* = *False*, *json:* *dict[str, Any]* | *None* = *None*)

The main notebook container class.

#### euporie.preview.tabs.notebook.Box

**class** euporie.preview.tabs.notebook.**Box**(*body: AnyContainer*, *padding: AnyDimension = None*, *padding_left: AnyDimension = None*, *padding_right: AnyDimension = None*, *padding_top: AnyDimension = None*, *padding_bottom: AnyDimension = None*, *width: AnyDimension = None*, *height: AnyDimension = None*, *style: str = ''*, *char: None | str | Callable[[], str] = None*, *modal: bool = False*, *key_bindings:* KeyBindings | *None* = *None*)

Add padding around a container.

This also makes sure that the parent can provide more space than required by the child. This is very useful when wrapping a small element with a fixed size into a `VSplit` or `HSplit` object. The `HSplit` and `VSplit` try to make sure to adapt respectively the width and height, possibly shrinking other elements. Wrapping something in a `Box` makes it flexible.

**Parameters**

- **body** – Another container object.
- **padding** – The margin to be used around the body. This can be
- **padding_left** (*overridden by*) – *padding_bottom*.
- **padding_right`** – *padding_bottom*.

- **and** (*padding_top*) – *padding_bottom*.
- **style** – A style string.
- **char** – Character to be used for filling the space around the body. (This is supposed to be a character with a terminal width of 1.)

### euporie.preview.tabs.notebook.Cell

**class** euporie.preview.tabs.notebook.**Cell**(*index: int*, *json: dict*, *kernel_tab:* BaseNotebook, *is_new: bool = False*)

A kernel_tab cell element.

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

### euporie.preview.tabs.notebook.ConditionalContainer

**class** euporie.preview.tabs.notebook.**ConditionalContainer**(*content: AnyContainer*, *filter: FilterOrBool*)

Wrapper around any other container that can change the visibility. The received *filter* determines whether the given container should be displayed or not.

**Parameters**

- **content** – *Container* instance.
- **filter** – *Filter* instance.

### euporie.preview.tabs.notebook.Dimension

**class** euporie.preview.tabs.notebook.**Dimension**(*min: int | None = None*, *max: int | None = None*, *weight: int | None = None*, *preferred: int | None = None*)

Specified dimension (width/height) of a user control or window.

The layout engine tries to honor the preferred size. If that is not possible, because the terminal is larger or smaller, it tries to keep in between min and max.

**Parameters**

- **min** – Minimum size.
- **max** – Maximum size.
- **weight** – For a VSplit/HSplit, the actual size will be determined by taking the proportion of weights from all the children. E.g. When there are two children, one with a weight of 1, and the other with a weight of 2, the second will always be twice as big as the first, if the min/max values allow it.
- **preferred** – Preferred size.

### euporie.preview.tabs.notebook.DynamicContainer

**class** euporie.preview.tabs.notebook.**DynamicContainer**(*get_container: Callable[[], AnyContainer]*)

Container class that dynamically returns any Container.

**Parameters**

**get_container** – Callable that returns a *Container* instance or any widget with a __pt_container__ method.

### euporie.preview.tabs.notebook.FastDictCache

**class** euporie.preview.tabs.notebook.**FastDictCache**(*get_value: Callable[[...], _V]*, *size: int = 1000000*)

Fast, lightweight cache which keeps at most *size* items. It will discard the oldest items in the cache first.

The cache is a dictionary, which doesn't keep track of access counts. It is perfect to cache little immutable objects which are not expensive to create, but where a dictionary lookup is still much faster than an object instantiation.

**Parameters**

**get_value** – Callable that's called in case of a missing key.

### euporie.preview.tabs.notebook.PreviewNotebook

**class** euporie.preview.tabs.notebook.**PreviewNotebook**(*app: BaseApp*, *path: Path | None = None*, *use_kernel_history: bool = False*)

A notebook tab which renders cells sequentially.

### euporie.preview.tabs.notebook.PrintingContainer

**class** euporie.preview.tabs.notebook.**PrintingContainer**(*children: Callable | Sequence[AnyContainer]*, *width: AnyDimension = None*, *key_bindings: KeyBindingsBase | None = None*)

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

### euporie.preview.tabs.notebook.VSplit

**class** euporie.preview.tabs.notebook.**VSplit**(*children: Sequence[AnyContainer]*, *window_too_small: Container | None = None*, *align: HorizontalAlign = HorizontalAlign.JUSTIFY*, *padding: AnyDimension = 0*, *padding_char: str | None = None*, *padding_style: str = ''*, *width: AnyDimension = None*, *height: AnyDimension = None*, *z_index: int | None = None*, *modal: bool = False*, *key_bindings: KeyBindingsBase | None = None*, *style: str | Callable[[], str] = ''*)

Several layouts, one stacked left/right of the other.

### euporie.preview.tabs.notebook.Window

**class** euporie.preview.tabs.notebook.**Window**(*content: UIControl | None = None, width: AnyDimension = None, height: AnyDimension = None, z_index: int | None = None, dont_extend_width: FilterOrBool = False, dont_extend_height: FilterOrBool = False, ignore_content_width: FilterOrBool = False, ignore_content_height: FilterOrBool = False, left_margins: Sequence[Margin] | None = None, right_margins: Sequence[Margin] | None = None, scroll_offsets: ScrollOffsets | None = None, allow_scroll_beyond_bottom: FilterOrBool = False, wrap_lines: FilterOrBool = False, get_vertical_scroll: Callable[[Window], int] | None = None, get_horizontal_scroll: Callable[[Window], int] | None = None, always_hide_cursor: FilterOrBool = False, cursorline: FilterOrBool = False, cursorcolumn: FilterOrBool = False, colorcolumns: None | list[ColorColumn] | Callable[[], list[ColorColumn]] = None, align: WindowAlign | Callable[[], WindowAlign] = WindowAlign.LEFT, style: str | Callable[[], str] = '', char: None | str | Callable[[], str] = None, get_line_prefix: GetLinePrefixCallable | None = None*)

Container that holds a control.

**class** euporie.preview.tabs.notebook.**PreviewNotebook**(*app: BaseApp, path: Path | None = None, use_kernel_history: bool = False*)

Bases: *BaseNotebook*

A notebook tab which renders cells sequentially.

**after_render**(*app: Application[Any]*) → None

Close the tab if all cells have been rendered.

**allow_stdin: bool = False**

**before_render**(*app: Application[Any]*) → None

Run the cell before rendering it if needed.

**bg_init = False**

**property cell: *Cell***

Return the current cell.

**change_kernel**(*msg: str | None = None, startup: bool = False*) → None

Prompt the user to select a new kernel.

**close**(*cb: Callable | None = None*) → None

Clean up render hooks before the tab is closed.

**comm_close**(*content: dict, buffers: Sequence[bytes]*) → None

Close a notebook Comm.

**comm_msg**(*content: dict, buffers: Sequence[bytes]*) → None

Respond to a Comm message from the kernel.

**comm_open**(*content: dict*, *buffers: Sequence[bytes]*) → None
Register a new kernel Comm object in the notebook.

**comms: dict[str, *Comm*]**

**completers: list[*Completer*]**

**container: AnyContainer**

**property current_input: *KernelInput***
Return the currently active kernel input, if any.

**default_callbacks: *MsgCallbacks***

**edit_mode = False**

**file_extensions: ClassVar[dict[str, None]] = {}**

**focus**() → None
Focus the tab (or make it visible).

**formatters: list[*Formatter*]**

**get_cell**(*index: int*) → *Cell*
Render a cell by its index.

**get_cell_by_id**(*cell_id: str*) → *euporie.core.widgets.cell.Cell* | None
Return a reference to the *Cell* container with a given cell id.

**history: *History***

**init_kernel**(*kernel:* Kernel | *None* = *None*, *comms: dict[str,* Comm*]* | *None* = *None*, *use_kernel_history: bool* = *False*, *connection_file:* Path | *None* = *None*) → None
Set up the tab's kernel and related components.

**inspectors: list[*Inspector*]**

**interrupt_kernel**() → None
Interrupt the current *Notebook*'s kernel.

**kernel: *Kernel***

**kernel_died**() → None
Call if the kernel dies.

**property kernel_display_name: str**
Return the display name of the kernel defined in the notebook JSON.

**property kernel_lang_file_ext: str**
Return the display name of the kernel defined in the notebook JSON.

**kernel_language: str**

**property kernel_name: str**
Return the name of the kernel defined in the notebook JSON.

**kernel_started**(*result: dict* | *None* = *None*) → None
Resume rendering the app when the kernel has started.

**lang_file_ext**() → str
Return the file extension for scripts in the notebook's language.

**property language: str**
Return the name of the kernel defined in the notebook JSON.

**load**() → None
Load the notebook file from the file-system.

**load_container**() → AnyContainer
Load the notebook's main container.

**async load_history**() → None
Load kernel history.

**async load_lsps**() → None
We do not need LSP support for preview notebook.

**load_widgets_from_metadata**() → None
Load widgets from state saved in notebook metadata.

**lsp_after_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we saved a document.

**lsp_before_save_handler**(*lsp:* LspClient) → None
Tell the the LSP we are about to save a document.

**lsp_change_handler**(*lsp:* LspClient) → None
Tell the LSP server a file metadata has changed.

**lsp_close_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_open_handler**(*lsp:* LspClient) → None
Tell the LSP we opened a file.

**lsp_update_diagnostics**(*lsp:* LspClient) → None
Process a new diagnostic report from the LSP.

**lsps: list[*LspClient*]**

**property metadata: dict[str, Any]**
Return a dictionary to hold notebook / kernel metadata.

**mime_types: ClassVar[set[str]] = {}**

**multiple_cells_selected: *Filter***

**name: str | None = None**

**property path_name: str**
Return the path name.

**post_init_kernel**() → None
Optionally start kernel after it is loaded.

**pre_init_kernel**() → None
Filter cells before kernel is loaded.

**print_title**() → None
Print a notebook's filename.

**refresh**(*slice_: slice | None = None*, *scroll: bool = False*) → None
Refresh the notebook.

**refresh_cell**(*cell:* Cell) → None
Trigger the refresh of a notebook cell.

**rendered_cells**() → list[*euporie.core.widgets.cell.Cell*]
Return a list of rendered notebooks' cells.

**report**() → *Report*
Return the current diagnostic reports.

**report_kernel_error**(*error: Exception | None*) → None
Report a kernel error to the user.

**reports: *WeakKeyDictionary*[*LspClient*, *Report*]**

**reset**() → None
Reload the notebook file from the disk and re-render.

**restart_kernel**(*cb: Callable | None = None*) → None
Restart the current *Notebook*'s kernel.

**run_cell**(*cell:* Cell, *wait: bool = False*, *callback: Callable[..., None] | None = None*) → None
Run a cell.

**Parameters**

- **cell** – The rendered cell to run. If `None`, runs the currently selected cell.
- **wait** – If `True`, blocks until cell execution is finished
- **callback** – Function to run after completion

**save**(*path:* Path | *None = None*, *cb: Callable | None = None*) → None
Write the notebook's JSON to the current notebook's file.

Additionally save the widget state to the notebook metadata.

**Parameters**

- **path** – An optional new path at which to save the tab
- **cb** – A callback to run if after saving the notebook.

**scroll_to**(*index: int*) → None
Scroll to a cell by index.

**select**(*cell_index: int*, *extend: bool = False*, *position: int | None = None*, *scroll: bool = False*) → None
Select a cell.

**property selected_indices: list[int]**
Return a list of the currently selected cell indices.

**set_kernel_info**(*info: dict*) → None
Handle kernel info requests.

**set_status** (*status: str*) → None
Call when kernel status changes.

**suggester: *AutoSuggest***

**property title: str**
Return the tab title.

**weight: int = 0**

CHAPTER

SIX

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## e

# INDEX

## Symbols

## A

## B

## C

## D

## E

## F

## G

## H

## I

## J



## K

## L

## M

## N

## O

## P

## Q



## R

## S

## T

## U

## V

## W

## X



## Y



## Z