psyplot GUI Documentation

Release 1.1.0

Philipp Sommer

Apr 09, 2018

Contents

Welcome! This package enhances the interactive visualization framework psyplot with a graphical user interface using PyQt. See the homepage of the main project on examples on the possibilities with psyplot.

CHAPTER 1

Documentation

1.1 Installation

This package requires the psyplot package which is installed alongside if you use `conda` or `pip`. However see the psyplot documentation for further informations.

1.1.1 How to install

Installation using conda

We highly recommend to use conda for installing psyplot-gui.

You can then install psyplot-gui simply via:

```
$ conda install -c conda-forge psyplot-gui
```

If you do not want to use PyQt4 (we indeed recommend to use PyQt5), you should remove the `'pyqt'` and and `'qt'` package from anaconda:

```
$ conda remove -y pyqt qt
```

You then have to install PyQt5 manually (see the installation page) or use an inofficial anaconda channel, e.g. the spyder-ide:

```
$ conda install -c spyder-ide pyqt5
```

Installation using pip

If you do not want to use conda for managing your python packages, you can also use the python package manager `pip` and install via:

```
$ pip install psyplot-gui
```

1.1.2 Dependencies

Required dependencies

Psyplot has been tested for python 2.7 and 3.4. Furthermore the package is built upon multiple other packages, namely

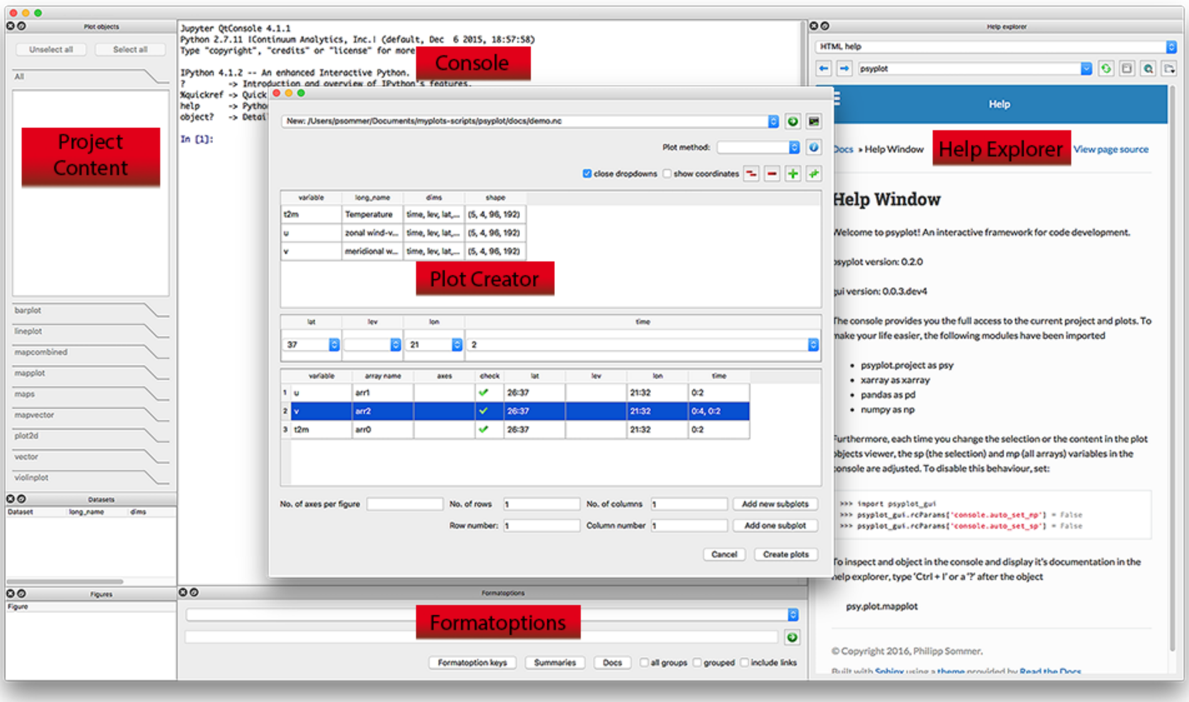
- psyplot>=0.2: The underlying framework for data visualization
- qtconsole>=4.1.1: A package providing the necessary objects for running an inprocess ipython console in a Qt widget
- fasteners: Which provides an inprocess lock to communicate to the psyplot mainwindow
- PyQt4 or PyQt5: Python bindings to the Qt software

Optional dependencies

We furthermore recommend to use

- sphinx>=1.3.5: To use all features of the interactive documentation access

1.2 Getting started



```
Jupyter QtConsole 4.1.1
Python 2.7.11 |Continuum Analytics, Inc.| (default, Dec  6 2015, 18:57:58)
Type "copyright", "credits" or "license" for more

IPython 4.1.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick
help      -> Pytho
object?   -> Detai

In [1]:
```

New: /Users/psommer/Documents/myplots-scripts/psyplot/docs/demo.nc

variable	long_name	dims	shape
t2m	Temperature	time, lev, lat,...	(5, 4, 96, 192)
u	zonal wind-v...	time, lev, lat,...	(5, 4, 96, 192)
v	meridional w...	time, lev, lat,...	(5, 4, 96, 192)

lat	lev	lon	time
37		21	2

	variable	array name	axes	check	lat	lev	lon	time
1	u	arr1		✓	26:37		21:32	0:2
2	v	arr2		✓	26:37		21:32	0:4, 0:2
3	t2m	arr0		✓	26:37		21:32	0:2

No. of rows 1
No. of columns 1
Row number: 1
Column number 1

Help Window

Welcome to psyplot! An interactive framework for code development.

psyplot version: 0.2.0

gui version: 0.0.3.dev4

The console provides you the full access to the current project and plots. To make your life easier, the following modules have been imported

- psyplot.project as psy
- xarray as xarray
- pandas as pd
- numpy as np

Furthermore, each time you change the selection or the content in the plot objects viewer, the sp (the selection) and mp (all arrays) variables in the console are adjusted. To disable this behaviour, set:

```
>>> import psyplot_gui
>>> psyplot_gui.rcParams['console.auto_set_mp'] = False
>>> psyplot_gui.rcParams['console.auto_set_sp'] = False
```

To inspect and object in the console and display it's documentation in the help explorer, type 'Ctrl + I' or a '?' after the object

psy.plot.mapplot

© Copyright 2016, Philipp Sommer.

Built with Sphinx using a theme provided by Read the Docs

The Screenshot above shows you the essential parts of the GUI:

- *The Console*: An IPython console

- *The Help Explorer*: A browser to display help and browse in the internet
- *The Plot Creator*: A widget to create new plots and open datasets
- *The Project content*: A widget to interact with the psyplot project
- *The formatoptions widget*: A widget to update and change formatoptions

1.2.1 Starting the GUI

Starting the GUI is straight forward but depends on how you installed it. If you installed it via *conda* or *pip*, just open a terminal (or Command Window `cmd` on Windows) and type `psyplot`. If you installed it through the standalone-installers (see Installation via standalone installers) and decided to add the binaries to your `PATH` variable (the default setting), just type `psyplot` in the terminal/cmd.

Otherwise, on MacOSX, look for the *Psyplot* app, e.g. via spotlight, and on Windows look in the *Start* → *All Programs* → *Psyplot* directory in your Windows start menu.

1.2.2 The Console

The central widget in the GUI is an in-process IPython console that provides the possibility to communicate with the psyplot package via the command line and to load any other module or to run any other script.

It is based on the qtconsole module and it is, by default, connected to the *help explorer*. If you type, for example,

```
np.sum(
```

it will show you the documentation of the `numpy.sum()` module in the *help explorer*. The same comes, if you type

```
np.sum?
```

This feature is motivated from the Spyder editor and can be disabled via the rcParams key `console.connect_to_help` (see *Configuration of the GUI*) or the little IP: symbol at the top of the help explorer.

Furthermore, is connected to the current psyplot project (see `psyplot.project.scp()` and `psyplot.project.gcp()`). Those are

sp This variable links to the current subproject (`psy.gcp()`)

mp This variable links to the current main project (`psy.gcp(True)`)

The following example, which you can just copy and paste in the console of the GUI, illustrates this:

```
# in the beginning, sp and mp are empty
In [1]: print(sp)
   ...: print(mp)
   ...:
psyplot.project.Project([
])
1 Main psyplot.project.Project([
])

In [3]: psy.plot.lineplot(xr.DataArray([1, 2, 3], name='test').to_dataset())
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\Out[3]: psyplot.
↪project.Project([arr0: psyplot.data.InteractiveList([    arr0: 1-dim DataArray of␣
↪test, with (dim_0)=(3,), ])])
```

(continues on next page)

(continued from previous page)

```
# now they both contain this data
In [4]: print(sp)
   ...: print(mp)
   ...:
psyplot.project.Project([arr0: psyplot.data.InteractiveList([    arr0: 1-dim␣
↪DataArray of test, with (dim_0)=(3,), ])])
1 Main psyplot.project.Project([arr0: psyplot.data.InteractiveList([    arr0: 1-dim␣
↪DataArray of test, with (dim_0)=(3,), ])])

# lets create a second one
In [6]: psy.plot.lineplot(xr.DataArray([1, 2, 3], name='test2').to_dataset())
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪psyplot.project.Project([arr1: psyplot.data.InteractiveList([    arr0: 1-dim␣
↪DataArray of test2, with (dim_0)=(3,), ])])

# sp now only contains the new one, mp contains both
In [7]: print(sp)
   ...: print(mp)
   ...:
psyplot.project.Project([arr1: psyplot.data.InteractiveList([    arr0: 1-dim␣
↪DataArray of test2, with (dim_0)=(3,), ])])
1 Main psyplot.project.Project([
    arr0: psyplot.data.InteractiveList([        arr0: 1-dim DataArray of test, with␣
↪(dim_0)=(3,), ]),
    arr1: psyplot.data.InteractiveList([        arr0: 1-dim DataArray of test2, with␣
↪(dim_0)=(3,), ])])
```

If you do not wish this behaviour, set the `console.auto_set_mp` and `console.auto_set_sp` rcParams to `False`.

1.2.3 The Help Explorer

The help explorer provides you access to python objects and online information. It can be used as a webbrowser if the icon is not clicked or the `help_explorer.online` rcParams key is True (see *Configuration of the GUI*).

It is motivated by the Help of the Spyder editor and uses Sphinx to automatically render python documentation written in restructured Text.

By default, the help explorer uses the `intersphinx` extension to link to different online libraries. This however always consumes time at the startup and can be disabled by the `help_explorer.use_intersphinx` rcParams key.

It is also connected to the information functions of psyplot, e.g. the `psyplot.plotter.Plotter.show_keys()` function. For example

```
In [9]: psy.plot.lineplot.keys()
+----------------+----------------+----------------+----------------+
| axiscolor      | color          | coord          | error          |
+----------------+----------------+----------------+----------------+
| erroralpha     | figtitle       | figtitleprops  | figtitlesize   |
+----------------+----------------+----------------+----------------+
| figtitleweight | grid           | labelprops     | labelsize      |
+----------------+----------------+----------------+----------------+
| labelweight    | legend         | legendlabels   | linewidth      |
+----------------+----------------+----------------+----------------+
```

(continues on next page)

(continued from previous page)

```
| marker          | markersize      | maskbetween     | maskgeq         |
+-----------------+-----------------+-----------------+-----------------+
| maskgreater     | maskleq         | maskless        | plot            |
+-----------------+-----------------+-----------------+-----------------+
| post            | post_timing     | sym_lims        | text            |
+-----------------+-----------------+-----------------+-----------------+
| ticksize        | tickweight      | tight           | title           |
+-----------------+-----------------+-----------------+-----------------+
| titleprops      | titlesize       | titleweight     | transpose       |
+-----------------+-----------------+-----------------+-----------------+
| xlabel          | xlim            | xrotation       | xticklabels     |
+-----------------+-----------------+-----------------+-----------------+
| xtickprops      | xticks          | ylabel          | ylim            |
+-----------------+-----------------+-----------------+-----------------+
| yrotation       | yticklabels     | ytickprops      | yticks          |
+-----------------+-----------------+-----------------+-----------------+
```

would be converted to HTML and shown in the help explorer.

1.2.4 The Plot Creator

The plot creator is used to create new plots from a `xarray.Dataset`. You can open it via *File* → *New Plot*.

1.2.5 The Project content

The project content shows you the current project (see `psyplot.project.gcp()`). The selected arrays are the current subproject.

1.2.6 The formatoptions widget

The formatoption widget can be used to update the formatoptions of the current subproject or to show their help.

1.3 Configuration of the GUI

As psyplot is configured by the `psyplot.config.rcsetup.rcParams`, psyplot-gui is configured by the *`psyplot_gui.config.rcsetup.rcParams`* dictionary.

Both dictionaries can also be modified through the *Preferences* widget (on MacOS, `Command+,`, on Windows and Linux: *Help* → *Preferences*).

As for `psyplot`, the rcParams are stored in the psyplot configuration directory, which is, under Linux and OSX by default, located at `$HOME/.config/psyplot/psyplotguirc.yml` and under Windows at `$HOME/.psyplot/psyplotguirc.yml`. This file might look like

```
In [1]: from psyplot_gui import rcParams

In [2]: print(rcParams.dump())
# Configuration parameters of the psyplot module
#
# You can copy this file (or parts of it) to another path and save it as
# psyplotguirc.yml. The directory should then be stored in the PSYPLOTCONFIGDIR
```

(continues on next page)

(continued from previous page)

```
# environment variable.
#
# psyplot gui version: 1.1.0
#
# Created with python
#
# 3.6.5 | packaged by conda-forge | (default, Apr  6 2018, 13:39:56)
# [GCC 4.8.2 20140120 (Red Hat 4.8.2-15)]
#
#
# Backend to use when using the graphical user interface. The current backend is used␣
↪and no changes are made. Note that it is usually not possible to change the backend␣
↪after importing the psyplot.project module. The default backend embeds the figures␣
↪into the
backend: psyplot
# If True, then the 'mp' variable in the console is automatically set when the␣
↪current main project changes
console.auto_set_mp: true
# If True, then the 'sp' variable in the console is automatically set when the␣
↪current sub project changes
console.auto_set_sp: true
# Whether the console shall be connected to the help_explorer or not
console.connect_to_help: true
# Start the different channels of the KernelClient
console.start_channels: true
# If True, a lazy load is performed on the arrays and data sets and their string␣
↪representation is displayed as tool tip. This part of the data into memory. It is␣
↪recommended to set this to False for remote data.
content.load_tooltips: true
# If True, the formatoptions in the Formatoptions widget are sorted by their␣
↪formatoption key rather than by their name.
fmt.sort_by_key: true
# Switch that controls whether the online functions of the help explorer shall be␣
↪enabled. False implies that help_explorer.use_intersphinx is set to False
help_explorer.online: null
# Boolean whether the html docs are rendered in a separate process
help_explorer.render_docs_parallel: true
# Use the intersphinx extension and link to the online documentations of matplotlib,␣
↪pyplot, psyplot, numpy, etc. when converting rst docstrings. The inventories are␣
↪loaded when the first object is documented. If None, intersphinx is only used with␣
↪PyQt5
help_explorer.use_intersphinx: null
# If True and the psyplot gui is already running, new files are opened in that gui
main.listen_to_port: true
# The port number used when new files are opened
main.open_files_port: 30124
# The plugins to exclude from loading. Can be either 'all' to exclude all plugins or␣
↪a list like in 'plugins.include'.
plugins.exclude: []
# The plugins to load. Can be either None to load all that are not explicitly␣
↪excluded by the 'plugins.exclude' key or a list of plugins to include. List items␣
↪can be either module names, plugin names or the module name and widget via '<module_
↪name>:<widget>'
plugins.include: null
```

1.4 Command line usage

The `psyplot-gui` module extends the command line usage of the psyplot module. You can open one (or more) files in the graphical user interface simply via:

```
$ psyplot myfile.nc
```

By default, if the gui is already running, the file is opened in this gui unless you specify the `ni` option.

Load a dataset, make the plot and save the result to a file

```
usage: psyplot [-h] [-V] [-aV] [-lp] [-lpm] [-lds]
               [-n [variable_name [variable_name ...]]]
               [-d dim,val1[,val2[,...]] [dim,val1[,val2[,...]] ...]]
               [-pm {'combined', 'fldmean', 'density', 'plot2d', 'violinplot',
↪'barplot', 'vector', 'lineplot'}]
               [-o str or list of str] [-p str] [-engine str] [-fmt FILENAME]
               [-t] [-rc RC_FILE] [-e str] [--enable-post] [-sns str]
               [-op str] [-cd str]
               [-chname [project-variable,variable-to-use [project-variable,variable-
↪to-use ...]]]
               [-b [backend]] [-ni] [-rc-gui RC_GUI_FILE] [-inc str [str ...]]
               [-exc str [str ...]] [--offline] [-pwd str] [-s str] [-c str]
               [-a] [-lgp]
               [str [str ...]]
```

1.4.1 Positional Arguments

str	Either the filenames to show, or, if the *project* parameter is set, the a list of ,-separated filenames to make a mapping from the original filename to a new one Default: []

1.4.2 Named Arguments

-n, --name	The variable names to plot if the *output* parameter is set Default: []
-d, --dims	A mapping from coordinate names to integers if the *project* is not given
-pm, --plot-method	Possible choices: combined, fldmean, density, plot2d, violinplot, barplot, vector, lineplot The name of the plot_method to use
-p, --project	If set, the project located at the given file name is loaded
-engine	The engine to use for opening the dataset (see `psyplot.data.open_dataset()`)
-fmt, --formatoptions	The path to a yaml (`'.yml'` or `'.yaml'`) or pickle file defining a dictionary of formatoption that is applied to the data visualized by the chosen *plot_method*
-rc, --rc-file	The path to a yaml configuration file that can be used to update the `rcParams`

-e, --encoding	The encoding to use for loading the project. If None, it is automatically determined by pickle. Note: Set this to `'latin1'` if using a project created with python2 on python3.
--enable-post	Enable the `post` processing formatoption. If True/set, post processing scripts are enabled in the given *project*. Only set this if you are sure that you can trust the given project file because it may be a security vulnerability. Default: False
-sns, --seaborn-style	The name of the style of the seaborn package that can be used for the `seaborn.set_style()` function
-cd, --concat-dim	The concatenation dimension if multiple files in *fnames* are provided Default: "__infer_concat_dim__"
-chname	A mapping from variable names in the project to variable names in the datasets that should be used instead. Variable names should be separated by a comma. Default: {}
-a, --use-all	If True, use all variables. Note that this is the default if the *output* is specified and not *name* Default: False

1.4.3 Info options

Options that print informations and quit afterwards

-V, --version	show program's version number and exit
-aV, --all-versions	Print the versions of all plugins and requirements and exit
-lp, --list-plugins	Print the names of the plugins and exit
-lpm, --list-plot-methods	List the available plot methods and what they do
-lds, --list-datasets	List the used dataset names in the given *project*.
-lgp, --list-gui-plugins	Print the names of the GUI plugins and exit. Note that the displayed plugins are not affected by the *include-plugins* and *exclude-plugins* options

1.4.4 Output options

Options that only have an effect if the *-o* option is set.

-o, --output	If set, the data is loaded and the figures are saved to the specified filename and now graphical user interface is shown
-t, --tight	If True/set, it is tried to figure out the tight bbox of the figure and adjust the paper size of the *output* to it Default: False
-op, --output-project	The name of a project file to save the project to. This option has only an effect if the *output* option is set.

1.4.5 Gui options

Options specific to the graphical user interface

-b, --backend The backend to use. By default, the 'gui.backend' key in the *rcParams* dictionary is used. If used without options, the default matplotlib backend is used.

Default: False

-ni, --new-instance If True/set and the *output* parameter is not set, a new application is created

Default: False

-rc-gui, --rc-gui-file The path to a yaml configuration file that can be used to update the *rcParams*

-inc, --include-plugins The plugin widget to include. Can be either None to load all that are not explicitly excluded by *exclude_plugins* or a list of plugins to include. List items can be either module names, plugin names or the module name and widget via '<module_name>:<widget>'. Default: None

-exc, --exclude-plugins The plugin widgets to exclude. Can be either 'all' to exclude all plugins or a list like in *include_plugins*.. Default: []

Default: []

--offline If True/set, psyplot will be started in offline mode without intersphinx and remote access for the help explorer

Default: False

-pwd The path to the working directory to use. Note if you do not provide any *fnames* or *project*, but set the *pwd*, it will switch the *pwd* of the current GUI.

-s, --script The path to a python script that shall be run in the GUI. If the GUI is already running, the commands will be executed in this GUI.

-c, --command Python commands that shall be run in the GUI. If the GUI is already running, the commands will be executed in this GUI

Examples

Here are some examples on how to use psyplot from the command line.

Plot the variable 't2m' in a netCDF file 'myfile.nc' and save the plot to 'plot.pdf':

```
$ psyplot myfile.nc -n t2m -pm mapplot -o test.pdf
```

Create two plots for 't2m' with the first and second timestep on the second vertical level:

```
$ psyplot myfile.nc -n t2m  -pm mapplot -o test.pdf -d t,0,1 z,1
```

If you have save a project using the psyplot.project.Project.save_project() method into a file named 'project.pkl', you can replot this via:

```
$ psyplot -p project.pkl -o test.pdf
```

If you use a different dataset than the one you used in the project (e.g. 'other_ds.nc'), you can replace it via:

```
$ psyplot other_dataset.nc -p project.pkl -o test.pdf
```

or explicitly via:

```
$ psyplot old_ds.nc,other_ds.nc -p project.pkl -o test.pdf
```

You can also load formatoptions from a configuration file, e.g.:

```
$ echo 'title: my title' > fmt.yaml
$ psyplot myfile.nc -n t2m  -pm mapplot -fmt fmt.yaml -o test.pdf
```

If you omit the `'-o'` option, the file is opened in the graphical user interface and if you run:

```
$ psyplot -pwd .
```

It will switch the directory of the already running GUI (if existent) to the current working directory in your terminal. Additionally,:

```
$ psyplot -s myscript.py
```

will run the file `'myscript.py'` in the GUI and:

```
$ psyplot -c 'print("Hello World!")'
```

will execute `print("Hello World")` in the GUI. The output, of the *-s* and *-c* options, will, however, be shown in the terminal.

1.5 Plugin configuration

The psyplot GUI has several built-in plugins, e.g. the *`help_explorer`* or the *`fmt_widget`*. External libraries can *add plugins* and the user can disable or enable them with through the *configuration*.

Note: These plugins should only affect the GUI. For other plugins that define new plotmethods, etc., see the psyplot documentation

1.5.1 Plugin configuration

You can include and exclude plugins either through the `include-plugins` and `exclude-plugins` command line option (see *Command line usage*), or you do it permanently with the *`rcParams`* (see *Configuration of the GUI*).

1.5.2 Developing plugins

External libraries insert the GUI as an entry point. In the `setup.py` script of a package, include the following:

```
setup(...,
      entry_points={'psyplot_gui': [
                         'widget-name1=widget-module1:widget-class-name1',
                         'widget-name2=widget-module2:widget-class-name2',
                         ...],
                    },
      ...)
```

Here, *widget-name1* is an arbitrary name you want to assign to the widget, *widget-module1* is the module from where to import the plugin, and *widget-class-name1* is the name of the class that inherits the `psyplot_gui.common.DockMixin` class.

For the `help_explorer`, this, for example, would like like

```
setup(...,
      entry_points={'psyplot_gui': [
                        'help=psyplot_gui.help_explorer:HelpExplorer',
                        ],
                    },
      ...)
```

1.6 API Reference

Core package for the psyplot graphical user interface

Classes

`ListGuiPluginsAction`(option_strings[, dest, . . .])

Functions

`get_parser`([create])	Return a parser to make that can be used to make plots or open files
`get_versions`([requirements])	
`send_files_to_psyplot`(callback, fnames, . . .)	Simple socket client used to send the args passed to the psyplot executable to an already running instance.
`start_app`([fnames, name, dims, plot_method, . . .])	Eventually start the QApplication or only make a plot

class psyplot_gui.**ListGuiPluginsAction**(*option_strings*, *dest='==SUPPRESS=='*, *nargs=None*, *default='==SUPPRESS=='*, ***kwargs*)

Bases: `argparse.Action`

psyplot_gui.**get_parser**(*create=True*)

Return a parser to make that can be used to make plots or open files from the command line

Returns The `argparse.ArgumentParser` instance

Return type psyplot.parser.FuncArgParser

See also:

`psyplot.main.get_parser()`, `psyplot.parser.FuncArgParser()`, `psyplot.main.main()`

psyplot_gui.**get_versions**(*requirements=True*)

psyplot_gui.**send_files_to_psyplot**(*callback*, *fnames*, *project*, **args*)

Simple socket client used to send the args passed to the psyplot executable to an already running instance.

This function has to most parts been taken from spyder `psyplot_gui.`**`start_app`**(*fnames=[]*, *name=[]*, *dims=None*, *plot_method=None*, *output=None*, *project=None*, *engine=None*, *formatoptions=None*, *tight=False*, *encoding=None*, *enable_post=False*, *seaborn_style=None*, *output_project=None*, *concat_dim='__infer_concat_dim__'*, *chname={}*, *backend=False*, *new_instance=False*, *rc_file=None*, *rc_gui_file=None*, *include_plugins=None*, *exclude_plugins=[]*, *offline=False*, *pwd=None*, *script=None*, *command=None*, *exec_=True*, *use_all=False*, *callback=None*)

Eventually start the QApplication or only make a plot

Parameters

- **`fnames`** (*`list of str`*) – Either the filenames to show, or, if the *project* parameter is set, the a list of *,*-separated filenames to make a mapping from the original filename to a new one
- **`name`** (*`list of str`*) – The variable names to plot if the *output* parameter is set
- **`dims`** (*`dict`*) – A mapping from coordinate names to integers if the *project* is not given
- **`plot_method`** (*`str`*) – The name of the plot_method to use
- **`output`** (*`str or list of str`*) – If set, the data is loaded and the figures are saved to the specified filename and now graphical user interface is shown
- **`project`** (*`str`*) – If set, the project located at the given file name is loaded
- **`engine`** (*`str`*) – The engine to use for opening the dataset (see `psyplot.data.open_dataset()`)
- **`formatoptions`** (*`dict`*) – A dictionary of formatoption that is applied to the data visualized by the chosen *plot_method*
- **`tight`** (*`bool`*) – If True/set, it is tried to figure out the tight bbox of the figure and adjust the paper size of the *output* to it
- **`rc_file`** (*`str`*) – The path to a yaml configuration file that can be used to update the `rcParams`
- **`encoding`** (*`str`*) – The encoding to use for loading the project. If None, it is automatically determined by pickle. Note: Set this to `'latin1'` if using a project created with python2 on python3.
- **`enable_post`** (*`bool`*) – Enable the `post` processing formatoption. If True/set, post processing scripts are enabled in the given *project*. Only set this if you are sure that you can trust the given project file because it may be a security vulnerability.
- **`seaborn_style`** (*`str`*) – The name of the style of the seaborn package that can be used for the `seaborn.set_style()` function
- **`output_project`** (*`str`*) – The name of a project file to save the project to
- **`concat_dim`** (*`str`*) – The concatenation dimension if multiple files in *fnames* are provided
- **`chname`** (*`dict`*) – A mapping from variable names in the project to variable names in the datasets that should be used instead
- **`backend`** (*`None or str`*) – The backend to use. By default, the `'gui.backend'` key in the *`rcParams`* dictionary is used. Otherwise it can be None to use the standard matplotlib backend or a string identifying the backend
- **`new_instance`** (*`bool`*) – If True/set and the *output* parameter is not set, a new application is created

- **rc_gui_file** (*str*) – The path to a yaml configuration file that can be used to update the `rcParams`
- **include_plugins** (*list of str*) – The plugin widget to include. Can be either None to load all that are not explicitly excluded by *exclude_plugins* or a list of plugins to include. List items can be either module names, plugin names or the module name and widget via `'<module_name>:<widget>'`
- **exclude_plugins** (*list of str*) – The plugin widgets to exclude. Can be either `'all'` to exclude all plugins or a list like in *include_plugins*.
- **offline** (*bool*) – If True/set, psyplot will be started in offline mode without intersphinx and remote access for the help explorer
- **pwd** (*str*) – The path to the working directory to use. Note if you do not provide any *fnames* or *project*, but set the *pwd*, it will switch the *pwd* of the current GUI.
- **script** (*str*) – The path to a python script that shall be run in the GUI. If the GUI is already running, the commands will be executed in this GUI.
- **command** (*str*) – Python commands that shall be run in the GUI. If the GUI is already running, the commands will be executed in this GUI
- **use_all** (*bool*) – If True, use all variables. Note that this is the default if the *output* is specified and not *name*
- **exec** (*bool*) – If True, the main loop is entered.
- **callback** (*str*) – A unique identifier for the method that should be used if psyplot is already running. Set this parameter to None to avoid sending

Returns `None` if *exec_* is True, otherwise the created `MainWindow` instance

Return type None or `psyplot_gui.main.MainWindow`

1.6.1 Subpackages

psyplot_gui.compat package

Submodules

psyplot_gui.compat.qtcompat module

Compatibility module for the different versions of PyQt

Functions

`asstring`(s)	
`isstring`(s)	

`psyplot_gui.compat.qtcompat.`**`asstring`**`(s)`

`psyplot_gui.compat.qtcompat.`**`isstring`**`(s)`

psyplot_gui.config package

Configuration module of the psyplot package

This module contains the module for managing rc parameters and the logging. Default parameters are defined in the `rcsetup.defaultParams` dictionary, however you can set up your own configuration in a yaml file (see `psyplot.load_rc_from_file()`)

Data

`config_path`	*class* – *str* or `None`. Path to the yaml configuration file (if found).
`logcfg_path`	`str`. Path to the yaml logging configuration file

`psyplot_gui.config.`**`config_path = None`**

class – *str* or `None`. Path to the yaml configuration file (if found). See `psyplot.config.rcsetup.psyplot_fname()` for further information

`psyplot_gui.config.`**`logcfg_path = None`**

`str`. Path to the yaml logging configuration file

Submodules

psyplot_gui.config.rcsetup module

Default management of the psyplot_gui package

This module defines the necessary configuration parts for the psyplot gui

Classes

`GuiRcParams`(*args, **kwargs)	RcParams for the psyplot-gui package.

Data

`defaultParams`	`dict` with default values and validation functions
`rcParams`	`RcParams` instance that stores default

Functions

`try_and_error`(*funcs)	Apply multiple validation functions
`validate_all`(v)	Test if `v == 'all'`
`validate_none`(b)	Validate that None is given
`validate_str`(s)	Validate a string

`class` `psyplot_gui.config.rcsetup.`**`GuiRcParams`**`(*args, **kwargs)`

Bases: `psyplot.config.rcsetup.RcParams`

RcParams for the psyplot-gui package.

Parameters **`defaultParams`** (`dict`) – The defaultParams to use (see the `defaultParams` attribute). By default, the `psyplot.config.rcsetup.defaultParams` dictionary is used

Other Parameters **‘‘*args, **kwargs‘‘** – Any key-value pair for the initialization of the dictionary

Attributes

HEADER	str(object='') -> str

Methods

load_from_file([fname])	Update rcParams from user-defined settings
load_plugins(*args, **kwargs)	Load the plugins for the psyplot_gui MainWindow

HEADER = 'Configuration parameters of the psyplot module\n\nYou can copy this file (or

load_from_file(*fname=None*)
Update rcParams from user-defined settings

This function updates the instance with what is found in *fname*

Parameters fname (*str*) – Path to the yaml configuration file. Possible keys of the dictionary are defined by `config.rcsetup.defaultParams`. If None, the `config.rcsetup.psyplot_fname()` function is used.

See also:

`dump_to_file()`, `psyplot_fname()`

load_plugins(**args*, ***kwargs*)
Load the plugins for the psyplot_gui MainWindow

Returns A mapping from entry point name to the imported widget class

Return type dict

Notes

`*args` and `**kwargs` are ignored

`psyplot_gui.config.rcsetup.`**`defaultParams`**
`dict` with default values and validation functions

`psyplot_gui.config.rcsetup.`**`rcParams`**
`RcParams` instance that stores default formatoptions and configuration settings.

`psyplot_gui.config.rcsetup.`**`try_and_error`**(**funcs*)
Apply multiple validation functions

Parameters *funcs – Validation functions to test

Returns

Return type function

`psyplot_gui.config.rcsetup.`**`validate_all`**(*v*)
Test if `v == 'all'`

`psyplot_gui.config.rcsetup.`**`validate_none`**(*b*)
Validate that None is given

Parameters b (*{None, 'none'}*) – None or string (the case is ignored)

Returns

Return type None

Raises `ValueError`

psyplot_gui.config.rcsetup.**validate_str**(*s*)

Validate a string

Parameters **s** (*str*) –

Returns

Return type str

Raises ValueError

psyplot_gui.sphinx_supp package

Submodules

psyplot_gui.sphinx_supp.conf module

Functions

link_aliases(app, what, name, obj, options, …)	
setup(app)	

psyplot_gui.sphinx_supp.conf.**link_aliases**(*app*, *what*, *name*, *obj*, *options*, *lines*)

psyplot_gui.sphinx_supp.conf.**setup**(*app*)

1.6.2 Submodules

psyplot_gui.backend module

Matplotlib backend to include matplotlib figures as dockwidgets in the psyplot gui

This backend is based upon matplotlibs qt4agg and qt5agg backends.

Classes

FigureCanvas	The canvas class with reimplemented resizing
FigureManager	The canvas manager for the psyplot backend interacting with the
FigureWidget	A simple container for figures in the psyplot backend
FiguresDock	Reimplemented QDockWidget to remove the dock widget when closed
PsyplotCanvas(figure)	The canvas class with reimplemented resizing
PsyplotCanvasManager(canvas, num)	The canvas manager for the psyplot backend interacting with the

Functions

new_figure_manager(num, *args, **kwargs)	Create a new figure manager instance
new_figure_manager_given_figure(num, figure)	Create a new figure manager instance for the given figure.

resizeEvent(event)	Reimplemented to make sure that the figure is only resized for

psyplot_gui.backend.**FigureCanvas**
Methods

alias of *psyplot_gui.backend.PsyplotCanvas*

resize(width, height)	set the canvas size in pixels
statusBar(*args, **kwargs)	

psyplot_gui.backend.**FigureManager**
Methods

alias of *psyplot_gui.backend.PsyplotCanvasManager*

class psyplot_gui.backend.**FigureWidget**
Bases: *psyplot_gui.common.DockMixin*, PyQt5.QtWidgets.QWidget

A simple container for figures in the psyplot backend **Miscallaneous**

dock_cls	Reimplemented QDockWidget to remove the dock widget when closed

dock_cls
alias of *FiguresDock*

class psyplot_gui.backend.**FiguresDock**
Bases: PyQt5.QtWidgets.QDockWidget

Reimplemented QDockWidget to remove the dock widget when closed **Methods**

close(*args, **kwargs)	Reimplemented to remove the dock widget from the mainwindow when closed

close (**args*, ***kwargs*)
Reimplemented to remove the dock widget from the mainwindow when closed

class psyplot_gui.backend.**PsyplotCanvas** (*figure*)
Bases: matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg

The canvas class with reimplemented resizing **Methods**

resizeEvent(event)	Reimplemented to make sure that the figure is only resized for

resizeEvent (*event*)
Reimplemented to make sure that the figure is only resized for events with height and width greater 0

class psyplot_gui.backend.**PsyplotCanvasManager** (*canvas*, *num*)
Bases: matplotlib.backends.backend_qt5.FigureManagerQT

The canvas manager for the psyplot backend interacting with the mainwindow of the psyplot gui **Methods**

resize(width, height)	set the canvas size in pixels

Continued on next page

Table 18 – continued from previous page

statusBar(*args, **kwargs)

resize(*width*, *height*)
set the canvas size in pixels

statusBar(**args*, ***kwargs*)

toolbar = None

psyplot_gui.backend.**new_figure_manager**(*num*, **args*, ***kwargs*)
Create a new figure manager instance

psyplot_gui.backend.**new_figure_manager_given_figure**(*num*, *figure*)
Create a new figure manager instance for the given figure.

psyplot_gui.common module

Common functions used for the psyplot gui

Classes

DockMixin	A mixin class to define psyplot_gui plugins
ListValidator(valid[, sep])	A validator class to validate that a string consists of strings in a
LoadFromConsoleButton([instances])	A toolbutton to load an object from the console
PyErrorMessage	Widget designed to display python errors via the showTraceback()
StreamToLogger(logger[, log_level])	Fake file-like stream object that redirects writes to a logger instance.

Functions

get_icon(name)	Get the path to an icon in the icons directory
get_module_path(modname)	Return module *modname* base path

class psyplot_gui.common.**DockMixin**
Bases: object

A mixin class to define psyplot_gui plugins

Notes

Attributes

config_page	The config page for this widget.
dock	The instance of QDockWidget of this plugin
dock_position	The position of the plugin
hidden	Boolean that is True if the dock widget should be hidden automatically
is_shown	Boolean that is True, if the dock widget is shown
title	The title of the plugin

Miscallaneous

`dock_cls`	The class to use for the DockWidget

Methods

`hide_plugin`()	Hide the plugin widget
`show_plugin`()	Show the plugin widget
`show_status_message`(msg)	Show a status message
`to_dock`(main[, title, position, docktype])	

Each external plugin should set the `dock_position` and the `title` attribute!

`config_page = None`
The config page for this widget. Should inherit the `psyplot_gui.preferences.ConfigPage` widget

`dock = None`
The instance of `QDockWidget` of this plugin

`dock_cls`
alias of `PyQt5.QtWidgets.QDockWidget`

`dock_position = None`
The position of the plugin

`hidden = False`
Boolean that is True if the dock widget should be hidden automatically after startup

`hide_plugin`()
Hide the plugin widget

`is_shown`
Boolean that is True, if the dock widget is shown

`show_plugin`()
Show the plugin widget

`show_status_message`(*msg*)
Show a status message

`title = None`
The title of the plugin

`to_dock`(*main*, *title=None*, *position=None*, *docktype='pane'*, **args*, ***kwargs*)

class `psyplot_gui.common.`**`ListValidator`**(*valid*, *sep=', '*, **args*, ***kwargs*)
Bases: `PyQt5.QtGui.QRegExpValidator`

A validator class to validate that a string consists of strings in a list of strings

Parameters

- **valid** (`list of str`) – The possible choices
- **sep** (`str, optional`) – The separation pattern
- ***args, **kwargs** – Determined by PyQt5.QtGui.QValidator

class `psyplot_gui.common.`**`LoadFromConsoleButton`**(*instances=None*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QToolButton`

A toolbutton to load an object from the console

Parameters **instances** (*class or tuple of classes*) – The classes that should be used for an instance check

Methods

check(obj)	
get_from_shell([oname])	Open an input dialog, receive an object and emit the
get_obj(oname)	Load an object from the current shell

Attributes

instances2check_str	
object_loaded(*args, **kwargs)	The signal that is emitted when an object has been loaded.
potential_object_names	

check(*obj*)

get_from_shell(*oname=None*)
Open an input dialog, receive an object and emit the *object_loaded* signal

get_obj(*oname*)
Load an object from the current shell

instances2check_str

object_loaded(**args*, ***kwargs*)
The signal that is emitted when an object has been loaded. The first argument is the object name, the second the object itself

potential_object_names

class psyplot_gui.common.**PyErrorMessage**
Bases: PyQt5.QtWidgets.QErrorMessage

Widget designed to display python errors via the *showTraceback()* method **Methods**

excepthook(type, value, traceback)	
showTraceback([header])	

excepthook(*type*, *value*, *traceback*)

showTraceback(*header=None*)

class psyplot_gui.common.**StreamToLogger**(*logger*, *log_level=20*)
Bases: object

Fake file-like stream object that redirects writes to a logger instance. **Methods**

flush()	
write(buf)	

flush()

write(*buf*)

`psyplot_gui.common.`**`get_icon`**(*name*)
Get the path to an icon in the icons directory

`psyplot_gui.common.`**`get_module_path`**(*modname*)
Return module *modname* base path

psyplot_gui.console module

An example of opening up an RichJupyterWidget in a PyQT Application, this can execute either stand-alone or by importing this file and calling inprocess_qtconsole.show(). Based on the earlier example in the IPython repository, this has been updated to use qtconsole.

Classes

`ConsoleWidget`(main, *args, **kwargs)	A console widget to access an inprocess shell
`IPythonControl`	A modified control to show the help of objects in the help explorer

`class` `psyplot_gui.console.`**`ConsoleWidget`**(*main*, **args*, ***kwargs*)
Bases: `qtconsole.inprocess.QtInProcessRichJupyterWidget`

A console widget to access an inprocess shell

Parameters

- **`help_explorer`** (`psyplot_gui.help_explorer.HelpExplorer` *or* `None`) – A widget that can be used to show the documentation of an object
- **`*args, **kwargs`** – Any other keyword argument for the `qtconsole.rich_jupyter_widget.RichJupyterWidget`

Methods

`close`(self)	
`get_current_object`([to_end])	Get the name of the object at cursor position
`get_obj`(obj_text)	Get the object from the shell specified by *obj_text*
`run_command_in_shell`(command)	Run a script in the shell
`run_script_in_shell`(script)	Run a script in the shell
`show_current_help`([to_end, force])	Show the help of the object at the cursor position if
`update_mp`(project)	Update the *mp* variable in the shell is
`update_sp`(project)	Update the *sp* variable in the shell is

Miscallaneous

`custom_control`	A modified control to show the help of objects in the help explorer

Attributes

`intro_msg`	str(object='') -> str
`rc`	Class that keeps week reference to the base dictionary
`run_command`(*args, **kwargs)	
`run_script`(*args, **kwargs)	

close(*self*) → bool

custom_control
alias of `IPythonControl`

get_current_object(*to_end=False*)
Get the name of the object at cursor position

get_obj(*obj_text*)
Get the object from the shell specified by *obj_text*

Parameters **obj_text** (`str`) – The name of the variable as it is stored in the shell

Returns

- *bool* – True, if the object could be found
- *object or None* – The requested object or None if it could not be found

intro_msg = ''

rc = {'auto_set_mp': True, 'auto_set_sp': True, 'connect_to_help': True, 'start_cha

run_command(**args*, ***kwargs*)

run_command_in_shell(*command*)
Run a script in the shell

run_script(**args*, ***kwargs*)

run_script_in_shell(*script*)
Run a script in the shell

show_current_help(*to_end=False*, *force=False*)
Show the help of the object at the cursor position if `rcParams['console.connect_to_help']` is set

update_mp(*project*)
Update the *mp* variable in the shell is `rcParams['console.auto_set_mp']` with a main project

update_sp(*project*)
Update the *sp* variable in the shell is `rcParams['console.auto_set_sp']` with a sub project

class `psyplot_gui.console.`**IPythonControl**
Bases: `PyQt5.QtWidgets.QTextEdit`

A modified control to show the help of objects in the help explorer **Methods**

`keyPressEvent`(event)	Reimplement Qt Method - Basic keypress event handler

keyPressEvent(*event*)
Reimplement Qt Method - Basic keypress event handler

psyplot_gui.content_widget module

Module containing the project content widget to display the selection

This module redefines the `psyplot.project.Project` class with additional features for an interactive usage with graphical qt user interface. There is no need to import this module because the `GuiProject` class defined here replaces the project class in the `psyplot.project` module.

Classes

ArrayItem(ref, *args, **kwargs)	A listwidget item that takes it's informations from a given array
DatasetTree(*args, **kwargs)	A QTreeWidget showing informations on all datasets in the main project
DatasetTreeItem(ds[, columns])	A QTreeWidgetItem showing informations on one dataset in the main
FiguresTree(*args, **kwargs)	A tree widget sorting the arrays by their figure
FiguresTreeItem(ref, *args, **kwargs)	An item displaying the information on a data object in one figure
PlotterList([plotter_type])	QListWidget showing multiple ArrayItems of one Plotter class
ProjectContent(*args, **kwargs)	Display the content in the current project
ProjectContentWidget(*args, **kwargs)	A combination of selection buttons and the ProjectContent
SelectAllButton(*args, **kwargs)	A button to select all data objects in the current main project
SelectNoneButton(*args, **kwargs)	A button to select no data objects in the current main project

class psyplot_gui.content_widget.**ArrayItem**(*ref*, **args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QListWidgetItem`

A listwidget item that takes it's informations from a given array

Parameters

- **ref** (*weakref*) – The weak reference to the array to display
- ***args, **kwargs** – Are determined by the parent class

Attributes

arr	The `psyplot.data.InteractiveList` or

Methods

disconnect_from_array()	
set_text_from_array()	Set the text and tooltop from the

arr = None

The `psyplot.data.InteractiveList` or `psyplot.data.InteractiveArray` instance

disconnect_from_array()

set_text_from_array()

Set the text and tooltop from the `psyplot.data.InteractiveArray._short_info()` and __str__ methods

class psyplot_gui.content_widget.**DatasetTree**(**args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QTreeWidget`, *psyplot_gui.common.DockMixin*

A QTreeWidget showing informations on all datasets in the main project **Methods**

add_datasets_from_cp([project])	Clear the tree and add the datasets based upon the given *project*
create_dataset_tree()	Set up the columns and insert the *DatasetTreeItem*

Continued on next page

Table 36 – continued from previous page

make_plot(ds, name[, exec_])	
open_menu(pos)	
refresh_items([item])	
set_columns([columns])	Set up the columns in the DatasetTree.

Attributes

tooltips	dict() -> new empty dictionary

add_datasets_from_cp(*project=None*)
Clear the tree and add the datasets based upon the given *project*

Parameters project (*psyplot.project.Project*) – The project containing the data array. If the project is not a main project, it's main project is used.

create_dataset_tree()
Set up the columns and insert the *DatasetTreeItem* instances from the current project

make_plot(*ds*, *name*, *exec_=None*)

open_menu(*pos*)

refresh_items(*item=None*)

set_columns(*columns=['long_name', 'dims', 'shape']*)
Set up the columns in the DatasetTree.

Parameters columns (`list of str`) – A list of netCDF attributes that shall be shown in columns

tooltips = {'Add to project': 'Add this variable or a plot of it to the current proje

class psyplot_gui.content_widget.**DatasetTreeItem**(*ds*, *columns=[]*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QTreeWidgetItem`

A QTreeWidgetItem showing informations on one dataset in the main project **Methods**

add_variables([ds])	Add children of variables and coords to this TreeWidgetItem

add_variables(*ds=None*)
Add children of variables and coords to this TreeWidgetItem

class psyplot_gui.content_widget.**FiguresTree**(**args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QTreeWidget`, *psyplot_gui.common.DockMixin*

A tree widget sorting the arrays by their figure

This widget uses the current sub and main project to show the open figures **Methods**

add_figures_from_cp(project)	Add the items in this tree based upon the figures in the given

add_figures_from_cp(*project*)
Add the items in this tree based upon the figures in the given project

class psyplot_gui.content_widget.**FiguresTreeItem**(*ref*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QTreeWidgetItem`

An item displaying the information on a data object in one figure

Parameters **ref** (*weakref*) – The weak reference to the array containing the data

Methods

disconnect_from_array()	Disconect this item from the corresponding array
set_text_from_array()	Set the text and tooltop from the

disconnect_from_array()
Disconect this item from the corresponding array

set_text_from_array()
Set the text and tooltop from the `psyplot.data.InteractiveArray._short_info()` and __str__ methods

class psyplot_gui.content_widget.**PlotterList**(*plotter_type=None*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QListWidget`

QListWidget showing multiple ArrayItems of one Plotter class

Parameters

- **plotter_type** (*str or None*) – If str, it mus be an attribute name of the `psyplot.project.Project` class. Otherwise the full project is used
- ***args, **kwargs** – Are determined by the parent class

Attributes

array_items	Iterable of *ArrayItem* items in this list
arrays	List of The InteractiveBase instances in this list
can_import_plotter	bool(x) -> bool
is_empty	boolean. True if the current project does not contain any arrays in the
project_attribute	str. The name of the attribute of the `psyplot.project.Project`
updated_from_project(*args, **kwargs)	

Methods

disconnect_items()	Disconnect the items in this list from the arrays
update_cp(*args, **kwargs)	Update the current project from what is selected in this list
update_from_project(project)	Update the content from the given Project

Notes

When initialized, the content of the list is determined by `gcp(True)` and `gcp()`

array_items
Iterable of *ArrayItem* items in this list

arrays
List of The InteractiveBase instances in this list

can_import_plotter = True

disconnect_items()
Disconnect the items in this list from the arrays

is_empty = True
boolean. True if the current project does not contain any arrays in the attribute identified by the *project_attribute*

project_attribute = None
str. The name of the attribute of the psyplot.project.Project class

update_cp(**args*, ***kwargs*)
Update the current project from what is selected in this list

update_from_project(*project*)
Update the content from the given Project

Parameters **project** (*psyplot.project.Project*) – If the project is a main project, new items will be added. Otherwise only the current selection changes

updated_from_project(**args*, ***kwargs*)

class psyplot_gui.content_widget.**ProjectContent**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QToolBox

Display the content in the current project

This toolbox contains several *PlotterList* that show the content of the current main and subproject **Methods**

add_plotterlist(identifier[, force])	Create a *PlotterList* from an identifier from the
enable_list(list_widget)	Enable a given list widget based upon whether it is empty or not
update_current_list()	Update the current list from the current main and sub project
update_lists(p)	

Attributes

current_names	
lists	OrderedDict containing the *PlotterList* instances

add_plotterlist(*identifier*, *force=False*)
Create a *PlotterList* from an identifier from the psyplot.project.Project class

current_names

enable_list(*list_widget*)
Enable a given list widget based upon whether it is empty or not

lists = {}
OrderedDict containing the *PlotterList* instances of the different selection attributes

update_current_list()
Update the current list from the current main and sub project

update_lists(*p*)

class psyplot_gui.content_widget.**ProjectContentWidget**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QWidget, *psyplot_gui.common.DockMixin*

A combination of selection buttons and the ProjectContent

class `psyplot_gui.content_widget.`**`SelectAllButton`**(**args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QPushButton`

A button to select all data objects in the current main project **Methods**

`enable_from_project`(project)	Enable the button if the given project is not empty
`select_all`()	Select all arrays

`enable_from_project`(*project*)

Enable the button if the given project is not empty

`select_all`()

Select all arrays

class `psyplot_gui.content_widget.`**`SelectNoneButton`**(**args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QPushButton`

A button to select no data objects in the current main project **Methods**

`enable_from_project`(project)	Enable the button if the given project is not empty
`select_none`()	Clear current subproject

`enable_from_project`(*project*)

Enable the button if the given project is not empty

`select_none`()

Clear current subproject

psyplot_gui.dataframeeditor module

A widget to display and edit DataFrames

Classes

`DataFrameDock`	The QDockWidget for the :class:‘DataFrameEditor
`DataFrameEditor`(*args, **kwargs)	An editor for data frames
`DataFrameModel`(df[, parent, index_editable, ...])	DataFrame Table Model
`DataFrameView`(df, parent, *args, **kwargs)	Data Frame view class
`FrozenTableView`(parent)	This class implements a table with its first column frozen

Functions

`bool_false_check`(value)	Used to convert bool intrance to false since any string in bool(‘’)

class `psyplot_gui.dataframeeditor.`**`DataFrameDock`**

Bases: `PyQt5.QtWidgets.QDockWidget`

The QDockWidget for the :class:‘DataFrameEditor

Methods

close()	Reimplemented to remove the dock widget from the mainwindow when closed

close()
Reimplemented to remove the dock widget from the mainwindow when closed

class psyplot_gui.dataframeeditor.**DataFrameEditor**(**args*, ***kwargs*)
Bases: psyplot_gui.common.DockMixin, PyQt5.QtWidgets.QWidget

An editor for data frames **Attributes**

cell_edited(*args, **kwargs)	A signal that is emitted when a cell has been changed.
cleared(*args, **kwargs)	A signal that is emitted, if the table is cleared
hidden	bool(x) -> bool
rows_inserted(*args, **kwargs)	A signal that is emitted, if rows have been inserted into the dataframe.

Methods

clear_table()	Clear the table and emit the cleared signal
close(self)	
maybe_tabify()	
open_dataframe([fname])	Opens a file dialog and the dataset that has been inserted
set_df(df, *args, **kwargs)	Fill the table from a DataFrame
set_dtypes_changeable(state)	Set the DataFrameModel.dtypes_changeable attribute
set_index_editable(state)	Set the DataFrameModel.index_editable attribute
set_lbl_size_text([nrows, ncols])	Set the text of the lbl_size label to display the size
to_dock(main, *args, **kwargs)	
toggle_fmt_button(text)	
update_format()	Update the format of the table
update_index_editable()	

Miscallaneous

dock_cls	The QDockWidget for the :class:'DataFrameEditor

cell_edited(**args*, ***kwargs*)
A signal that is emitted when a cell has been changed. The argument is a tuple of two integers and one float: the row index, the column index and the new value

clear_table()
Clear the table and emit the cleared signal

cleared(**args*, ***kwargs*)
A signal that is emitted, if the table is cleared

close(*self*) → bool

dock_cls
alias of DataFrameDock

hidden

bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

maybe_tabify()

open_dataframe(*fname=None*, **args*, ***kwargs*)
Opens a file dialog and the dataset that has been inserted

rows_inserted(**args*, ***kwargs*)
A signal that is emitted, if rows have been inserted into the dataframe. The first value is the integer of the (original) position of the row, the second one is the number of rows

set_df(*df*, **args*, ***kwargs*)
Fill the table from a `DataFrame`

Parameters

- **df** (*pandas.DataFrame*) – The data frame that will be shown by this *DataFrameModel* instance
- **index_editable** (*bool*) – True if the index should be modifiable by the user
- **dtypes_changeable** (*bool*) – True, if the data types should be modifiable by the user
- **show** (*bool*) – If True (default), show and **raise_** the editor

set_dtypes_changeable(*state*)
Set the `DataFrameModel.dtypes_changeable` attribute

set_index_editable(*state*)
Set the `DataFrameModel.index_editable` attribute

set_lbl_size_text(*nrows=None*, *ncols=None*)
Set the text of the `lbl_size` label to display the size

to_dock(*main*, **args*, ***kwargs*)

toggle_fmt_button(*text*)

update_format()
Update the format of the table

update_index_editable()

class psyplot_gui.dataframeeditor.**DataFrameModel**(*df*, *parent=None*, *index_editable=True*, *dtypes_changeable=True*)

Bases: `PyQt5.QtCore.QAbstractTableModel`

DataFrame Table Model

Parameters

- **df** (*pandas.DataFrame*) – The data frame that will be shown by this *DataFrameModel* instance
- **parent** (`DataFrameEditor`) – The editor for the table
- **index_editable** (*bool*) – True if the index should be modifiable by the user
- **dtypes_changeable** (*bool*) – True, if the data types should be modifiable by the user

Attributes

`COLS_TO_LOAD`	int(x=0) -> integer
`ROWS_TO_LOAD`	int(x=0) -> integer

Methods

`bgcolor`(state)	Toggle backgroundcolor
`can_fetch_more`([rows, columns])	
`columnCount`([index])	DataFrame column number
`data`(index[, role])	Cell content
`fetch_more`([rows, columns])	
`flags`(index)	Set flags
`get_format`()	Return current format
`get_value`(row, column)	Returns the value of the DataFrame
`headerData`(section, orientation[, role])	Set header data
`insertRow`(irow)	Insert one row into the `df`
`insertRows`(irow[, nrows])	Insert a row into the `df`
`reset`()	
`rowCount`([index])	DataFrame row number
`setData`(index, value[, role, change_type])	Cell content change
`set_format`(format)	Change display format
`sort`(column[, order, return_check, report])	Overriding sort method
`update_df_index`()	"Update the DataFrame index

COLS_TO_LOAD = 40

ROWS_TO_LOAD = 500

bgcolor (*state*)
Toggle backgroundcolor

can_fetch_more (*rows=False*, *columns=False*)

columnCount (*index=<PyQt5.QtCore.QModelIndex object>*)
DataFrame column number

data (*index*, *role=0*)
Cell content

fetch_more (*rows=False*, *columns=False*)

flags (*index*)
Set flags

get_format ()
Return current format

get_value (*row*, *column*)
Returns the value of the DataFrame

headerData (*section*, *orientation*, *role=0*)
Set header data

insertRow (*irow*)
Insert one row into the `df`

Parameters **irow** (*int*) – The row index. If iRow is equal to the length of the df, the new row will be appended.

insertRows (*irow*, *nrows=1*)
Insert a row into the df

Parameters

- **irow** (*int*) – The row index. If *irow* is equal to the length of the df, the rows will be appended.
- **nrows** (*int*) – The number of rows to insert

reset ()

rowCount (*index=<PyQt5.QtCore.QModelIndex object>*)
DataFrame row number

setData (*index*, *value*, *role=2*, *change_type=None*)
Cell content change

set_format (*format*)
Change display format

sort (*column*, *order=0*, *return_check=False*, *report=True*)
Overriding sort method

update_df_index ()
"Update the DataFrame index

class psyplot_gui.dataframeeditor.**DataFrameView** (*df*, *parent*, **args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QTableView

Data Frame view class

Parameters

- **df** (*pandas.DataFrame*) – The data frame that will be shown by this DataFrameModel instance
- **parent** (DataFrameEditor) – The editor for the table
- **index_editable** (*bool*) – True if the index should be modifiable by the user
- **dtypes_changeable** (*bool*) – True, if the data types should be modifiable by the user

Methods

change_type(func)	A function that changes types of cells
contextMenuEvent(event)	Reimplement Qt method
copy()	Copy text to clipboard
insert_row_above_selection()	Insert rows above the selection
insert_row_below_selection()	Insert rows below the selection
load_more_data(value[, rows, columns])	
moveCursor(cursor_action, modifiers)	Update the table position.
reset_model()	
resizeEvent(event)	Update the frozen column dimensions.
scrollTo(index, hint)	Scroll the table.
set_df(df, *args, **kwargs)	Set the DataFrame for this table
set_index([append])	Set the index from the selected columns
setup_menu()	Setup context menu

Continued on next page

Table 55 – continued from previous page

sortByColumn(index)	Implement a Column sort
update_section_height(logical_index, …)	Update the vertical width of the frozen column when a
update_section_width(logical_index, …)	Update the horizontal width of the frozen column when a

Attributes

filled	True if the table is filled with content

change_type (*func*)
A function that changes types of cells

contextMenuEvent (*event*)
Reimplement Qt method

copy ()
Copy text to clipboard

filled
True if the table is filled with content

insert_row_above_selection ()
Insert rows above the selection

The number of rows inserted depends on the number of selected rows

insert_row_below_selection ()
Insert rows below the selection

The number of rows inserted depends on the number of selected rows

load_more_data (*value*, *rows=False*, *columns=False*)

moveCursor (*cursor_action*, *modifiers*)
Update the table position.

Updates the position along with the frozen column when the cursor (selector) changes its position

reset_model ()

resizeEvent (*event*)
Update the frozen column dimensions.

Updates takes place when the enclosing window of this table reports a dimension change

scrollTo (*index*, *hint*)
Scroll the table.

It is necessary to ensure that the item at index is visible. The view will try to position the item according to the given hint. This method does not takes effect only if the frozen column is scrolled.

set_df (*df*, **args*, ***kwargs*)
Set the `DataFrame` for this table

Parameters

- **df** (*pandas.DataFrame*) – The data frame that will be shown by this *DataFrameModel* instance
- **index_editable** (*bool*) – True if the index should be modifiable by the user

- **dtypes_changeable** (*bool*) – True, if the data types should be modifiable by the user

set_index(*append=False*)
Set the index from the selected columns

setup_menu()
Setup context menu

sortByColumn(*index*)
Implement a Column sort

update_section_height(*logical_index*, *old_size*, *new_size*)
Update the vertical width of the frozen column when a change takes place on any of the rows

update_section_width(*logical_index*, *old_size*, *new_size*)
Update the horizontal width of the frozen column when a change takes place in the first column of the table

class psyplot_gui.dataframeeditor.**FrozenTableView**(*parent*)
Bases: `PyQt5.QtWidgets.QTableView`

This class implements a table with its first column frozen For more information please see: http://doc.qt.io/qt-5/qtwidgets-itemviews-frozencolumn-example.html

Constructor. **Methods**

contextMenuEvent(event)	Show the context Menu
update_geometry()	Update the frozen column size when an update occurs in its parent

contextMenuEvent(*event*)
Show the context Menu

Reimplemented to show the use the contextMenuEvent of the parent

update_geometry()
Update the frozen column size when an update occurs in its parent table

psyplot_gui.dataframeeditor.**bool_false_check**(*value*)
Used to convert bool intrance to false since any string in bool('') will return True

psyplot_gui.dependencies module

Dependencies widget of the psyplot package

This module defines the `DependenciesWidget` that shows the versions of of psyplot, psyplot_gui, psyplot plugins and their requirements

Classes

DependenciesDialog(versions, *args, **kwargs)	A dialog for displaying the dependencies
DependenciesTree(versions, *args, **kwargs)	A tree widget to display dependencies

class psyplot_gui.dependencies.**DependenciesDialog**(*versions*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QDialog`

A dialog for displaying the dependencies **Attributes**

bt_copy	The QPushButton used for copying selected packages to the clipboard
info_label	A label for simple status update
label	description label
timer	A QTimer that clears the `info_label` after some time
tree	The *DependenciesTree* that contains the package infos
vbox	the QVBoxLayout containing all the widgets

Methods

clear_label()	Clear the info label
copy_selected([label])	Copy the selected versions and items to the clipboard

bt_copy = None
    The QPushButton used for copying selected packages to the clipboard

clear_label()
    Clear the info label

copy_selected(*label=None*)
    Copy the selected versions and items to the clipboard

info_label = None
    A label for simple status update

label = None
    description label

timer = None
    A QTimer that clears the *info_label* after some time

tree = None
    The *DependenciesTree* that contains the package infos

vbox = None
    the QVBoxLayout containing all the widgets

class psyplot_gui.dependencies.**DependenciesTree**(*versions*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QTreeWidget`

A tree widget to display dependencies

This widget uses a dictionary as created through the `psyplot.get_versions()` function to display the requirements and versions.

Parameters versions (*dict*) – The dictionary that contains the version information

Methods

add_dependencies(versions[, parent])	Add the version informations to the tree
open_menu(position)	Open a menu to expand and collapse all items in the tree

See also:

`psyplot.get_versions`

add_dependencies(*versions*, *parent=None*)
    Add the version informations to the tree

This method creates an QTreeWidgetItem for each package in *versions* and adds it to this tree.

Parameters **parent** (*QTreeWidgetItem*) – The parent of the newly created items for the packages in *versions*. If None, the newly created items are inserted as top level items into the tree

open_menu(*position*)
Open a menu to expand and collapse all items in the tree

Parameters **position** (*QPosition*) – The position where to open the menu

psyplot_gui.fmt_widget module

Module defining a widget for updating the formatoption of the current project

Classes

DimensionsWidget(parent[, dim])	A widget for updating the dimensions
FormatoptionWidget(*args, **kwargs)	Widget to update the formatoptions of the current project

class psyplot_gui.fmt_widget.**DimensionsWidget**(*parent*, *dim=None*)
Bases: PyQt5.QtWidgets.QWidget

A widget for updating the dimensions **Methods**

get_ds()	
insert_from_combo()	
reset_combobox()	Clear all comboboxes
set_dim(dim)	
set_single_selection([yes])	
slice2list(sl)	
toggle_close_popup()	

get_ds()

insert_from_combo()

reset_combobox()
Clear all comboboxes

set_dim(*dim*)

set_single_selection(*yes=True*)

slice2list(*sl*)

toggle_close_popup()

class psyplot_gui.fmt_widget.**FormatoptionWidget**(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QWidget, *psyplot_gui.common.DockMixin*

Widget to update the formatoptions of the current project

This widget, mainly made out of a combobox for the formatoption group, a combobox for the formatoption, and a text editor, is designed for updating the selected formatoptions for the current subproject.

The widget is connected to the psyplot.project.Project.oncpchange signal and refills the comboboxes if the current subproject changes.

The text editor either accepts python code that will be executed by the given *console*, or yaml code.

Parameters

- **help_explorer** (psyplot_gui.help_explorer.HelpExplorer) – The help explorer to show the documentation of one formatoption
- **console** (psyplot_gui.console.ConsoleWidget) – The console that can be used to update the current subproject via:

  ```
  psy.gcp().update(**kwargs)
  ```

  where `**kwargs` is defined through the selected formatoption in the *fmt_combo* combobox and the value in the *line_edit* editor
- ****kwargs** (**args,*) – Any other keyword for the QWidget class

Methods

clear_text()	
fill_combos_from_project(project)	Fill `group_combo` and `fmt_combo` from a project
fill_fmt_combo(i[, current_text])	Fill the `fmt_combo` combobox based on the current group name
get_name(fmto)	Get the name of a `psyplot.plotter.Formatoption` instance
get_obj()	Get the current update text
get_text()	Get the current update text
insert_obj(obj)	Add a string to the formatoption widget
load_fmt_widget(i)	Load the formatoption specific widget
refill_from_rc(sort_by_key)	
remove_fmt_widget()	
reset_fmt_widget()	
run_code()	Run the update of the project inside the `shell`
set_current_fmt_value(i)	Add the value of the current formatoption to the line text
set_fmto(name)	
set_obj(obj)	
setup_fmt_completion_model()	
show_all_fmt_info(what)	Show the keys, summaries or docs of the formatoptions
show_fmt_info(i)	Show the documentation of the formatoption in the help explorer
toggle_line_edit()	Switch between the `line_edit` and `text_edit`

Attributes

fmt_combo	The combobox for the formatoptions
fmt_widget	The formatoption specific widget that is loaded from the formatoption
fmto	
group_combo	The combobox for the formatoption groups
help_explorer	The help_explorer to display the documentation of the formatoptions
line_edit	A line edit for updating the formatoptions
multiline_button	A button to switch between `line_edit` and `text_edit`
no_fmtos_update	update the fmto combo box or not

Continued on next page

Table 65 – continued from previous page

shell	The shell to execute the update of the formatoptions in the current
text_edit	A multiline text editor for updating the formatoptions

clear_text()

fill_combos_from_project(*project*)
Fill *group_combo* and *fmt_combo* from a project

Parameters **project** (*psyplot.project.Project*) – The project to use

fill_fmt_combo(*i*, *current_text=None*)
Fill the *fmt_combo* combobox based on the current group name

fmt_combo = None
The combobox for the formatoptions

fmt_widget = None
The formatoption specific widget that is loaded from the formatoption

fmto

get_name(*fmto*)
Get the name of a `psyplot.plotter.Formatoption` instance

get_obj()
Get the current update text

get_text()
Get the current update text

group_combo = None
The combobox for the formatoption groups

help_explorer = None
The help_explorer to display the documentation of the formatoptions

insert_obj(*obj*)
Add a string to the formatoption widget

line_edit = None
A line edit for updating the formatoptions

load_fmt_widget(*i*)
Load the formatoption specific widget

This method loads the formatoption specific widget from the `psyplot.plotter.Formatoption.get_fmt_widget()` method and displays it above the *line_edit*

Parameters **i** (*int*) – The index of the current formatoption

multiline_button = None
A button to switch between *line_edit* and *text_edit*

no_fmtos_update
update the fmto combo box or not

refill_from_rc(*sort_by_key*)

remove_fmt_widget()

reset_fmt_widget()

run_code()
Run the update of the project inside the *shell*

set_current_fmt_value(*i*)
Add the value of the current formatoption to the line text

set_fmto(*name*)

set_obj(*obj*)

setup_fmt_completion_model()

shell
The shell to execute the update of the formatoptions in the current project

show_all_fmt_info(*what*)
Show the keys, summaries or docs of the formatoptions

Calling this function let's the help browser show the documentation etc. of all docs or only the selected group determined by the state of the `grouped_cb` and `all_groups_cb` checkboxes

Parameters what (*{'keys', 'summaries', 'docs'}*) – Determines what to show

show_fmt_info(*i*)
Show the documentation of the formatoption in the help explorer

text_edit = None
A multiline text editor for updating the formatoptions

toggle_line_edit()
Switch between the *line_edit* and *text_edit*

This method is called when the *multiline_button* is clicked and switches between the single line :attr:``line_edit` and the multiline *text_edit*

psyplot_gui.help_explorer module

Help explorer widget supplying a simple web browser and a plain text help viewer

Classes

HelpExplorer(*args, **kwargs)	A widget for showing the documentation.
HelpMixin	Base class for providing help on an object
SphinxThread(outdir[, html_text_no_doc])	A thread to render sphinx documentation in a separate process
TextHelp(*args, **kwargs)	Class to show plain text rst docstrings
UrlBrowser(*args, **kwargs)	Very simple browser with session history and autocompletion based upon
UrlCombo(*args, **kwargs)	A editable ComboBox with autocompletion
UrlHelp(*args, **kwargs)	Class to convert rst docstrings to html and show browsers

Functions

file2html(fname)
html2file(url)

class psyplot_gui.help_explorer.**HelpExplorer**(**args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QWidget`, *psyplot_gui.common.DockMixin*

A widget for showing the documentation. It behaves somewhat similar to spyders object inspector plugin and can show restructured text either as html (if sphinx is installed) or as plain text. It furthermore has a browser to show html content

> **Warning:** The `HelpBrowser` class is known to crash under PyQt4 when new web page domains are loaded. Hence you should disable the browsing to different remote websites and even disable intersphinx

Methods

`close`(self)	
`set_viewer`(name)	Sets the current documentation viewer
`show_help`(obj[, oname, files])	Show the documentaion of the given object
`show_intro`([text])	Show an intro text
`show_rst`(text[, oname, files])	Show restructured text

Attributes

`viewers`	The viewer classes used by the help explorer.

`close`(*self*) → bool

`set_viewer`(*name*)

Sets the current documentation viewer

Parameters **`name`** (`str or object`) – A string must be one of the `viewers` attribute. An object can be one of the values in the `viewers` attribute

`show_help`(*obj*, *oname=''*, *files=None*)

Show the documentaion of the given object

We first try to use the current viewer based upon it's `HelpMixin.can_document_object` attribute. If this does not work, we check the other viewers

Parameters

- **`obj`** (`object`) – The object to get the documentation for
- **`oname`** (`str`) – The name to use for the object in the documentation
- **`files`** (`list of str`) – A path to additional files that shall be used to process show the docs

`show_intro`(*text=''*)

Show an intro text

We first try to use the current viewer based upon it's `HelpMixin.can_show_rst` attribute. If this does not work, we check the other viewers

Parameters **`s`** (`str`) – A string in reStructured Text format to show

`show_rst`(*text*, *oname=''*, *files=None*)

Show restructured text

We first try to use the current viewer based upon it's `HelpMixin.can_show_rst` attribute. If this does not work, we check the other viewers

Parameters

- **`text`** (`str`) – The text to show

- **oname** (*str*) – The object name
- **descriptor** (instance of `object_descriptor`) – The object descriptor holding the informations
- **files** (*list of str*) – A path to additional files that shall be used to display the docs

viewers = {'HTML help': <class 'psyplot_gui.help_explorer.UrlHelp'>, 'Plain text': <

The viewer classes used by the help explorer. *HelpExplorer* instances replace this attribute with the corresponding HelpMixin instance

class psyplot_gui.help_explorer.**HelpMixin**

Bases: `object`

Base class for providing help on an object **Attributes**

can_document_object	`bool` determining whether the documentation of an object can be
can_show_rst	`bool` determining whether this class can show restructured text

Methods

describe_object(obj[, oname])	Return an instance of the `object_descriptor` class
get_doc(descriptor)	Get the documentation of the object in the given *descriptor*
header(descriptor, sig)	Format the header and include object name and signature *sig*
process_docstring(lines, descriptor)	Make final modification on the rst lines
show_help(obj[, oname, files])	Show the rst documentation for the given object
show_intro([text])	Show an intro message
show_rst(text[, oname, descriptor, files])	Abstract method which needs to be implemented by th widget to show

Miscallaneous

object_descriptor	Object containing the necessary fields to describe an object given to the help widget.

can_document_object = True

`bool` determining whether the documentation of an object can be shown or not

can_show_rst = True

`bool` determining whether this class can show restructured text

describe_object(*obj*, *oname=''*)

Return an instance of the *object_descriptor* class

Returns The descriptor containing the information on the object

Return type *object_descriptor*

get_doc(*descriptor*)

Get the documentation of the object in the given *descriptor*

Parameters descriptor (instance of *object_descriptor*) – The descriptor containig the information on the specific object

Returns The header and documentation of the object in the descriptor

Return type str

Notes

This method uses the `IPython.core.oinspect.getdoc()` function to get the documentation and the `IPython.core.oinspect.signature()` function to get the signature. Those function (different from the inspect module) do not fail when the object is not saved

header(*descriptor*, *sig*)
Format the header and include object name and signature *sig*

Returns The header for the documentation

Return type str

object_descriptor
Object containing the necessary fields to describe an object given to the help widget. The descriptor is set up by the `describe_object()` method.

alias of `ObjectDescriptor`

process_docstring(*lines*, *descriptor*)
Make final modification on the rst lines

Returns The docstring

Return type str

show_help(*obj*, *oname=''*, *files=None*)
Show the rst documentation for the given object

Parameters

- **obj** (`object`) – The object to get the documentation for
- **oname** (`str`) – The name to use for the object in the documentation
- **files** (`list of str`) – A path to additional files that shall be used to process show the docs

show_intro(*text=''*)
Show an intro message

Parameters **s** (`str`) – A string in reStructured Text format to show

show_rst(*text*, *oname=''*, *descriptor=None*, *files=None*)
Abstract method which needs to be implemented by th widget to show restructured text

Parameters

- **text** (`str`) – The text to show
- **oname** (`str`) – The object name
- **descriptor** (instance of `object_descriptor`) – The object descriptor holding the informations
- **files** (`list of str`) – A path to additional files that shall be used to display the docs

Returns True if the text is displayed

Return type bool

class psyplot_gui.help_explorer.**SphinxThread**(*outdir*, *html_text_no_doc=''*)

Bases: `PyQt5.QtCore.QThread`

A thread to render sphinx documentation in a separate process **Attributes**

html_error(*args, **kwargs)	
html_ready(*args, **kwargs)	A signal to be emitted when the rendering finished.

Methods

render(doc, name)	Render the given rst string and save the file as `name + '.rst'`
run()	Create the html file.

html_error(**args*, ***kwargs*)

html_ready(**args*, ***kwargs*)

A signal to be emitted when the rendering finished. The url is the file location

render(*doc*, *name*)

Render the given rst string and save the file as `name + '.rst'`

Parameters

- **doc** (*str*) – The rst docstring
- **name** (*str*) – the name to use for the file

run()

Create the html file. When called the first time, it may take a while because the `sphinx.application.Sphinx` app is build, potentially with intersphinx

When finished, the html_ready signal is emitted

class psyplot_gui.help_explorer.**TextHelp**(**args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QFrame`, *psyplot_gui.help_explorer.HelpMixin*

Class to show plain text rst docstrings **Attributes**

None	The `PyQt5.QtWidgets.QPlainTextEdit` instance used for

Methods

show_rst(text, *args, **kwargs)	Show the given text in the editor window

editor = None

The `PyQt5.QtWidgets.QPlainTextEdit` instance used for displaying the documentation

show_rst(*text*, **args*, ***kwargs*)

Show the given text in the editor window

Parameters

- **text** (*str*) – The text to show
- ***args, **kwargs** – Are ignored

class psyplot_gui.help_explorer.**UrlBrowser**(**args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QFrame`

Very simple browser with session history and autocompletion based upon the `PyQt5.QtWebEngineWidgets.QWebEngineView` class

Warning: This class is known to crash under PyQt4 when new web page domains are loaded. Hence it should be handled with care

Methods

browse(url)	Make a web browse on the given url and show the page on the Webview widget.
toogle_lock()	Disable (or enable) the changing of the current webpage
toogle_url_lock()	Disable (or enable) the loading of web pages in www
update_url_lock_from_rc(online)	
url_changed(url)	Triggered when the url is changed to update the adress line

Attributes

bt_ahead	button to go to next url
bt_back	button to go to previous url
bt_lock	button to go lock to the current url
bt_refresh	refresh the current url
bt_url_lock	button to disable browsing in www
button_box	The upper part of the browser containing all the buttons
completed	Boolean whether the html page loading is completed.
default_url	The initial url showed in the webview.
doc_urls	Dictionary that remembers insertion order
None	The actual widget showing the html content
tb_url	adress line
url_like_re	Compiled regular expression objects
vbox	The upper most layout aranging the button box and the html widget

browse(*url*)

Make a web browse on the given url and show the page on the Webview widget.

bt_ahead = None

button to go to next url

bt_back = None

button to go to previous url

bt_lock = None

button to go lock to the current url

bt_refresh = None

refresh the current url

bt_url_lock = None

button to disable browsing in www

button_box = None
The upper part of the browser containing all the buttons

completed
Boolean whether the html page loading is completed.

default_url = None
The initial url showed in the webview. If None, nothing will be displayed

doc_urls = {'cartopy': 'http://scitools.org.uk/cartopy/docs/latest/index.html', 'numpy

html = None
The actual widget showing the html content

tb_url = None
adress line

toogle_lock()
Disable (or enable) the changing of the current webpage

toogle_url_lock()
Disable (or enable) the loading of web pages in www

update_url_lock_from_rc(*online*)

url_changed(*url*)
Triggered when the url is changed to update the adress line

url_like_re = re.compile('^\\\\w+://')

vbox = None
The upper most layout aranging the button box and the html widget

class psyplot_gui.help_explorer.**UrlCombo**(**args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QComboBox`

A editable ComboBox with autocompletion **Methods**

add_text_on_top([text, block])	Add the given text as the first item
keyPressEvent(event)	Handle key press events
setModel(model)	Reimplemented to also set the model of the filter and completer

add_text_on_top(*text=None*, *block=False*)
Add the given text as the first item

keyPressEvent(*event*)
Handle key press events

setModel(*model*)
Reimplemented to also set the model of the filter and completer

class psyplot_gui.help_explorer.**UrlHelp**(**args*, ***kwargs*)
Bases: *psyplot_gui.help_explorer.UrlBrowser*, *psyplot_gui.help_explorer.HelpMixin*

Class to convert rst docstrings to html and show browsers **Methods**

browse(url)	Reimplemented to add file paths to the url string
close(self)	

Continued on next page

Table 80 – continued from previous page

describe_object(obj[, oname])	Describe an object using additionaly the object type from the
get_doc(descriptor)	Reimplemented to (potentially) use the features from
get_objtype(obj)	Get the object type of the given object and determine wheter the
header(descriptor, sig)	Format the header and include object name and signature *sig*
is_importable(modname)	Determine whether members of the given module can be documented with
process_docstring(lines, descriptor)	Process the lines with the napoleon sphinx extension
reset_sphinx(value)	Method that is called if the configuration changes
show_help(obj[, oname, files])	Render the rst docu for the given object with sphinx and show it
show_intro([text])	Show the intro text in the explorer
show_rst(text[, oname, descriptor, files])	Render restructured text with sphinx and show it
toogle_connect_console()	Disable (or enable) the loading of web pages in www
toogle_url_lock()	Disable (or enable) the loading of web pages in www
update_connect_console(connect)	
url_changed(url)	Reimplemented to remove file paths from the url string

Attributes

bt_url_menus	menu button with different urls
can_document_object	bool(x) -> bool
can_show_rst	bool(x) -> bool
sphinx_thread	

Miscallaneous

object_descriptor	Object containing the necessary fields to describe an object given to the help widget.

browse(*url*)
Reimplemented to add file paths to the url string

bt_url_menus = None
menu button with different urls

can_document_object = True

can_show_rst = True

close(*self*) → bool

describe_object(*obj*, *oname=''*)
Describe an object using additionaly the object type from the `get_objtype()` method

Returns The descriptor of the object

Return type instance of `object_descriptor`

get_doc(*descriptor*)
Reimplemented to (potentially) use the features from sphinx.ext.autodoc

get_objtype(*obj*)

Get the object type of the given object and determine wheter the object is considered a class, a module, a function, method or data

Parameters **obj** (*object*) –

Returns One out of {'class', 'module', 'function', 'method', 'data'}

Return type str

header(*descriptor*, *sig*)
Format the header and include object name and signature *sig*

Returns The header for the documentation

Return type str

is_importable(*modname*)
Determine whether members of the given module can be documented with sphinx by using the `sphinx.util.get_module_source()` function

Parameters **modname** (*str*) – The __name__ attribute of the module to import

Returns True if sphinx can import the module

Return type bool

object_descriptor
Object containing the necessary fields to describe an object given to the help widget. The descriptor is set up by the *describe_object()* method and contains an additional objtype attribute

alias of `ObjectDescriptor`

process_docstring(*lines*, *descriptor*)
Process the lines with the napoleon sphinx extension

reset_sphinx(*value*)
Method that is called if the configuration changes

show_help(*obj*, *oname=''*, *files=None*)
Render the rst docu for the given object with sphinx and show it

Parameters

- **obj** (*object*) – The object to get the documentation for
- **oname** (*str*) – The name to use for the object in the documentation
- **files** (*list of str*) – A path to additional files that shall be used to process show the docs

show_intro(*text=''*)
Show the intro text in the explorer

Parameters **s** (*str*) – A string in reStructured Text format to show

show_rst(*text*, *oname=''*, *descriptor=None*, *files=None*)
Render restructured text with sphinx and show it

Parameters **%(HelpMixin.show_rst.parameters)s** –

sphinx_thread = None

toogle_connect_console()
Disable (or enable) the loading of web pages in www

toogle_url_lock()
Disable (or enable) the loading of web pages in www

update_connect_console(*connect*)

url_changed(*url*)
Reimplemented to remove file paths from the url string

psyplot_gui.help_explorer.**file2html**(*fname*)

psyplot_gui.help_explorer.**html2file**(*url*)

psyplot_gui.main module

Core module for the psyplot graphical user interface

This module redefines the `psyplot.project.Project` class with additional features for an interactive usage with graphical qt user interface. There is no need to import this module because the `GuiProject` class defined here replaces the project class in the `psyplot.project` module.

Classes

`MainWindow`([show])	**param show** If True, the created mainwindow is show

Data

`mainwindow`	The `PyQt5.QtWidgets.QMainWindow` of the graphical user interface

class psyplot_gui.main.**MainWindow**(*show=True*)
Bases: `PyQt5.QtWidgets.QMainWindow`

Parameters **show** (`bool`) – If True, the created mainwindow is show

Methods

`about`()	About the tool
`addDockWidget`(area, dockwidget[, docktype])	Reimplemented to add widgets to the windows menu
`add_mp_to_menu`()	
`change_cwd`(path)	Change the current working directory
`close`(self)	
`closeEvent`(event)	closeEvent reimplementation
`edit_preferences`([exec_])	Edit Spyder preferences
`eventually_add_mp_to_menu`(p)	
`excepthook`(type, value, traceback)	A method to replace the sys.excepthook
`export_mp`(*args, **kwargs)	
`export_sp`(*args, **kwargs)	
`focus_on_console`(*args, **kwargs)	Put focus on the ipython console
`new_data_frame_editor`([df, title])	Open a new dataframe editor
`new_plots`([exec_])	
`open_external_files`([fnames, project, . . .])	Open external files
`open_mp`(*args, **kwargs)	Open a new main project
`open_sp`(*args, **kwargs)	Open a subproject and add it to the current main project
`register_shortcut`(action, shortcut[, context])	Register an action for a shortcut

Continued on next page

Table 85 – continued from previous page

reset_rcParams()	
run([fnames, project, engine, plot_method, . . .])	Create a mainwindow and open the given files or project
run_app(*args, **kwargs)	Create a QApplication, open the given files or project and enter the
save_mp(*args, **kwargs)	Save the current main project
save_sp(*args, **kwargs)	Save the current sub project
setup_default_layout()	Set up the default window layout
show_dependencies([exec_])	Open a dialog that shows the dependencies
start_open_files_server()	This method listens to the open_files_port and opens the plot
update_project_action(num)	

Attributes

console	Inprocess console
current_shortcuts	The current keyboard shortcuts
dataframeeditors	the DataFrameEditor widgets
default_shortcuts	The keyboard shortcuts of the default layout
default_widths	default widths of the dock widgets
dockwidgets	The dockwidgets of this instance
ds_tree	tree widget displaying the open datasets
figures	list of figures from the psyplot backend
figures_tree	tree widget displaying the open figures
fmt_widget	general formatoptions widget
help_explorer	help explorer
logger	The logger of this instance
open_external(*args, **kwargs)	A signal that is emmitted when the a signal is received through the
open_files_server	The server to open external files
project_content	tab widget displaying the arrays in current main and sub project

about()
About the tool

addDockWidget(*area*, *dockwidget*, *docktype=None*, **args*, ***kwargs*)
Reimplemented to add widgets to the windows menu

add_mp_to_menu()

change_cwd(*path*)
Change the current working directory

close(*self*) → bool

closeEvent(*event*)
closeEvent reimplementation

console = None
Inprocess console

current_shortcuts = []
The current keyboard shortcuts

dataframeeditors = None
: the DataFrameEditor widgets

default_shortcuts = []
: The keyboard shortcuts of the default layout

default_widths = {}
: default widths of the dock widgets

dockwidgets = []
: The dockwidgets of this instance

ds_tree = None
: tree widget displaying the open datasets

edit_preferences(*exec_=None*)
: Edit Spyder preferences

eventually_add_mp_to_menu(*p*)

excepthook(*type*, *value*, *traceback*)
: A method to replace the sys.excepthook

export_mp(**args*, ***kwargs*)

export_sp(**args*, ***kwargs*)

figures = []
: list of figures from the psyplot backend

figures_tree = None
: tree widget displaying the open figures

fmt_widget = None
: general formatoptions widget

focus_on_console(**args*, ***kwargs*)
: Put focus on the ipython console

help_explorer = None
: help explorer

logger
: The logger of this instance

new_data_frame_editor(*df=None*, *title='DataFrame Editor'*)
: Open a new dataframe editor

Parameters

- **df** (`pandas.DataFrame`) – The dataframe to display
- **title** (`str`) – The title of the dock window

Returns The newly created editor

Return type *psyplot_gui.dataframeeditor.DataFrameEditor*

new_plots(*exec_=None*)

open_external(**args*, ***kwargs*)
: A signal that is emmitted when the a signal is received through the open_files_server

open_external_files (*fnames=[]*, *project=None*, *engine=None*, *plot_method=None*, *name=None*, *dims=None*, *encoding=None*, *enable_post=False*, *seaborn_style=None*, *concat_dim='__infer_concat_dim__'*, *chname={}*)

Open external files

Parameters

- **fnames** (`list of str`) – Either the filenames to show, or, if the *project* parameter is set, the a list of ,-separated filenames to make a mapping from the original filename to a new one
- **name** (`list of str`) – The variable names to plot if the *output* parameter is set
- **dims** (`dict`) – A mapping from coordinate names to integers if the *project* is not given
- **plot_method** (`str`) – The name of the plot_method to use
- **project** (`str`) – If set, the project located at the given file name is loaded
- **engine** (`str`) – The engine to use for opening the dataset (see `psyplot.data.open_dataset()`)
- **encoding** (`str`) – The encoding to use for loading the project. If None, it is automatically determined by pickle. Note: Set this to `'latin1'` if using a project created with python2 on python3.
- **enable_post** (`bool`) – Enable the `post` processing formatoption. If True/set, post processing scripts are enabled in the given *project*. Only set this if you are sure that you can trust the given project file because it may be a security vulnerability.
- **seaborn_style** (`str`) – The name of the style of the seaborn package that can be used for the `seaborn.set_style()` function
- **concat_dim** (`str`) – The concatenation dimension if multiple files in *fnames* are provided
- **chname** (`dict`) – A mapping from variable names in the project to variable names in the datasets that should be used instead

open_files_server = None

The server to open external files

open_mp (**args*, ***kwargs*)

Open a new main project

open_sp (**args*, ***kwargs*)

Open a subproject and add it to the current main project

project_content = None

tab widget displaying the arrays in current main and sub project tree widget displaying the open datasets

register_shortcut (*action*, *shortcut*, *context=2*)

Register an action for a shortcut

reset_rcParams ()

classmethod run (*fnames=[]*, *project=None*, *engine=None*, *plot_method=None*, *name=None*, *dims=None*, *encoding=None*, *enable_post=False*, *seaborn_style=None*, *concat_dim='__infer_concat_dim__'*, *chname={}*, *show=True*)

Create a mainwindow and open the given files or project

This class method creates a new mainwindow instance and sets the global `mainwindow` variable.

Parameters

- **fnames** (*list of str*) – Either the filenames to show, or, if the *project* parameter is set, the a list of `,`-separated filenames to make a mapping from the original filename to a new one
- **name** (*list of str*) – The variable names to plot if the *output* parameter is set
- **dims** (*dict*) – A mapping from coordinate names to integers if the *project* is not given
- **plot_method** (*str*) – The name of the plot_method to use
- **project** (*str*) – If set, the project located at the given file name is loaded
- **engine** (*str*) – The engine to use for opening the dataset (see `psyplot.data.open_dataset()`)
- **encoding** (*str*) – The encoding to use for loading the project. If None, it is automatically determined by pickle. Note: Set this to `'latin1'` if using a project created with python2 on python3.
- **enable_post** (*bool*) – Enable the `post` processing formatoption. If True/set, post processing scripts are enabled in the given *project*. Only set this if you are sure that you can trust the given project file because it may be a security vulnerability.
- **seaborn_style** (*str*) – The name of the style of the seaborn package that can be used for the `seaborn.set_style()` function
- **concat_dim** (*str*) – The concatenation dimension if multiple files in *fnames* are provided
- **chname** (*dict*) – A mapping from variable names in the project to variable names in the datasets that should be used instead
- **show** (*bool*) – If True, the created mainwindow is show

Notes

- There can be only one mainwindow at the time
- This method does not create a QApplication instance! See `run_app()`

See also:

`run_app()`

classmethod run_app(**args*, ***kwargs*)

Create a QApplication, open the given files or project and enter the mainloop

Parameters

- **fnames** (*list of str*) – Either the filenames to show, or, if the *project* parameter is set, the a list of `,`-separated filenames to make a mapping from the original filename to a new one
- **name** (*list of str*) – The variable names to plot if the *output* parameter is set
- **dims** (*dict*) – A mapping from coordinate names to integers if the *project* is not given
- **plot_method** (*str*) – The name of the plot_method to use
- **project** (*str*) – If set, the project located at the given file name is loaded
- **engine** (*str*) – The engine to use for opening the dataset (see `psyplot.data.open_dataset()`)

- **encoding** (*str*) – The encoding to use for loading the project. If None, it is automatically determined by pickle. Note: Set this to `'latin1'` if using a project created with python2 on python3.
- **enable_post** (*bool*) – Enable the `post` processing formatoption. If True/set, post processing scripts are enabled in the given *project*. Only set this if you are sure that you can trust the given project file because it may be a security vulnerability.
- **seaborn_style** (*str*) – The name of the style of the seaborn package that can be used for the `seaborn.set_style()` function
- **concat_dim** (*str*) – The concatenation dimension if multiple files in *fnames* are provided
- **chname** (*dict*) – A mapping from variable names in the project to variable names in the datasets that should be used instead
- **show** (*bool*) – If True, the created mainwindow is show

See also:

`run()`

save_mp(**args*, ***kwargs*)
Save the current main project

save_sp(**args*, ***kwargs*)
Save the current sub project

setup_default_layout()
Set up the default window layout

show_dependencies(*exec_=None*)
Open a dialog that shows the dependencies

start_open_files_server()
This method listens to the open_files_port and opens the plot creator for new files

This method is inspired and to most parts copied from spyder

update_project_action(*num*)

psyplot_gui.main.**mainwindow = None**
The `PyQt5.QtWidgets.QMainWindow` of the graphical user interface

psyplot_gui.plot_creator module

This module contains a widget to create new plots with psyplot

The main class is the `PlotCreator` which is used to handle the different plotting methods of the `psyplot.project.ProjectPlotter` class

Classes

`ArrayNameItemDelegate`	Delegate using the `ArrayNameValidator` for validation
`ArrayNameValidator`(text, table, *args, **kwargs)	Class to make sure that only those arrays names are inserted that are
`ArrayTable`(get_func[, columns])	Table that shows the arrays that will be used for plotting
`AxesCreator`([fig, x0, y0, x1, y1])	Widget to setup an axes in a arbitrary location

Continued on next page

Table 87 – continued from previous page

AxesCreatorCollection([key, func_kwargs])	Wrapper for a QToolBox that holds the different possibilities to select
AxesSelector(*args, **kwargs)	Widget to select an already created axes
AxesViewer(*args, **kwargs)	Widget to show a rectangle
CoordComboBox(ds_func, dim[, parent])	Combobox showing coordinate information of a dataset
CoordsTable(get_func, *args, **kwargs)	A table showing the coordinates of in a dataset via instances of
DragDropTable(*args, **kwargs)	Table that allows to exchange rows via drag and drop
PlotCreator(*args, **kwargs)	Widget to extract data from a dataset and eventually create a plot
SubplotCreator([fig, rows, cols, num1, num2])	Select a subplot to which will be created (if not already existing) when
VariableItemDelegate	Delegate alowing only the variables in the parents dataset.
VariablesTable(get_func[, columns])	Table to display the variables of a dataset

class psyplot_gui.plot_creator.**ArrayNameItemDelegate**
Bases: PyQt5.QtWidgets.QStyledItemDelegate

Delegate using the ArrayNameValidator for validation **Methods**

createEditor(self, QWidget, . . .)

createEditor (*self*, *QWidget*, *QStyleOptionViewItem*, *QModelIndex*) → QWidget

class psyplot_gui.plot_creator.**ArrayNameValidator** (*text*, *table*, **args*, ***kwargs*)
Bases: PyQt5.QtGui.QValidator

Class to make sure that only those arrays names are inserted that are not currently in the main project or the tree
Methods

fixup(self, str)
validate(self, str, int)

fixup (*self*, *str*) → str

validate (*self*, *str*, *int*) → Tuple[QValidator.State, str, int]

class psyplot_gui.plot_creator.**ArrayTable** (*get_func*, *columns=[]*, **args*, ***kwargs*)
Bases: psyplot_gui.plot_creator.DragDropTable

Table that shows the arrays that will be used for plotting

It contains the following columns:

1. The variable column which holds the variable names of the arrays. multiple variables may be separated by ';;'
2. The array name. The psyplot.data.InteractiveBase.arr_name attribute. Depending on the plot methods _prefer_list, multiple array names are allowed or not. If this attribute is True, arrays with the same array name will be concatenated into one psyplot.data.InteractiveList
3. The axes column. Use the right-click context menu to select a subplot
4. The check column. Checks for variable names, array names, axes and dimensions via the psyplot.project._PlotterInterface.check_data() method
5. Columns containing the dimension informations

Attributes

DIMS_TT	Base tool tip for a dimension column
VARIABLE_TT	Tool tip for the variable column
arr_col	The index of the array name column
arr_names_dict	The final dictionary containing the array names necessary for the
axes	A list of axes settings corresponding to the arrays in the
axes_col	The index of the axes column
axes_patt	pattern to interprete arbitrary axes
check_col	The index of the check column
current_names	The names that are currently in use
prefer_list	Return the _prefer_list attribute of the plot_method
sep	The separator for variable names
subplot_patt	Pattern to interprete subplots
var_col	The index of the variable column
vnames	The list of variable names per array

Methods

add_single_subplot(rows, cols, row, col)	Add one subplot to the selected arrays on multiple figures
add_subplots(rows, cols[, maxn])	Add multiple subplots to the selected arrays
axes_creator_action(rows)	Action to open a *AxesCreatorCollection* for the selected
axes_info(s)	Interpretes an axes information
check_array(row[, ignore_duplicates])	check whether the array variables are valid, the array name is
check_arrays(**kwargs)	Convenience function to check all arrays using the
check_item(item)	Check the array corresponding to the given item
dropEvent(event)	Reimplemented to call the `check_arrays()` after the call
get_all_rows(row)	Return all the rows that have the same array name as the given *row*
insert_array(name[, check])	Appends the settings for an array the the list in a new row
next_available_name(*args, **kwargs)	Gives the next possible name to use
remove_arrays([selected])	Remove array rows from the list
set_columns(columns)	Set the columns of the table
set_pm(s)	Set the plot method
setup_from_ds([ds, plot_method])	Fill the table based upon the given dataset.
showAxesCreator(pos)	Context menu for right-click on a row
update_other_items(item)	Updates the axes information of the other items corresponding
update_selected([check, dims])	Updates the dimensions of the selectiond arrays with the given

Parameters

- **`get_func`** (*`function`*) – The function that, when called without arguments, returns the xarray.Dataset to use
- **`columns`** (*`list of str`*) – The coordinates in the dataset

DIMS_TT = "The values for dimension %s. You can use integers either explicit, e.g.<ul>
Base tool tip for a dimension column

VARIABLE_TT = "The variables of the array from the dataset. Multiplevariables for one
Tool tip for the variable column

add_single_subplot(*rows*, *cols*, *row*, *col*)
Add one subplot to the selected arrays on multiple figures

add_subplots(*rows*, *cols*, *maxn=None*)
Add multiple subplots to the selected arrays

arr_col
The index of the array name column

arr_names_dict
The final dictionary containing the array names necessary for the *arr_names* parameter in the `psyplot.data.ArrayList.from_dataset()` method

axes
A list of axes settings corresponding to the arrays in the `arr_names_dict`

axes_col
The index of the axes column

axes_creator_action(*rows*)
Action to open a `AxesCreatorCollection` for the selected rows

axes_info(*s*)
Interpretes an axes information

axes_patt = re.compile('\\((?P<fig>\\d+),\\s*(?P<x0>0*\\.\\d+),\\s*(?P<y0>0*\\.\\d+),\\
pattern to interprete arbitrary axes

check_array(*row*, *ignore_duplicates=[]*)
check whether the array variables are valid, the array name is valid, the axes info is valid and the dimensions

check_arrays(***kwargs*)
Convenience function to check all arrays using the `check_array()` method

check_col
The index of the check column

check_item(*item*)
Check the array corresponding to the given item

current_names
The names that are currently in use

dropEvent(*event*)
Reimplemented to call the `check_arrays()` after the call

get_all_rows(*row*)
Return all the rows that have the same array name as the given *row*

insert_array(*name*, *check=True*, ***kwargs*)
Appends the settings for an array the the list in a new row

next_available_name(**args*, ***kwargs*)
Gives the next possible name to use

prefer_list
Return the _prefer_list attribute of the plot_method

remove_arrays(*selected=True*)
Remove array rows from the list

Parameters **selected** (*bool*) – If True, only the selected rows are removed

sep = ';;'
The separator for variable names

set_columns(*columns*)
Set the columns of the table

Parameters **columns** (*list of str*) – The coordinates in the dataset

set_pm(*s*)
Set the plot method

setup_from_ds(*ds=None*, *plot_method=None*)
Fill the table based upon the given dataset.

Parameters

- **ds** (*xarray.Dataset or None*) – If None, the dataset from the `get_ds` function is used
- **plot_method** (*psyplot.project._PlotterInterface or None*) – The plot method of the `psyplot.project.ProjectPlotter` class or None if no plot shall be made

showAxesCreator(*pos*)
Context menu for right-click on a row

subplot_patt = re.compile('\\((?P<fig>\\d+),\\s*(?P<rows>\\d+),\\s*(?P<cols>\\d+),\\s*
Pattern to interprete subplots

update_other_items(*item*)
Updates the axes information of the other items corresponding that have the same array name as the array corresponding to the given *item*

update_selected(*check=True*, *dims={}*)
Updates the dimensions of the selectiond arrays with the given *dims*

Parameters

- **check** (*bool*) – whether the array shall be checked afterwards
- **dims** (*dict*) – a mapping from coordinate names to string values that shall be appended to the current text

var_col
The index of the variable column

vnames
The list of variable names per array

class psyplot_gui.plot_creator.**AxesCreator**(*fig=None*, *x0=0.125*, *y0=0.1*, *x1=0.9*, *y1=0.9*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QWidget`

Widget to setup an axes in a arbitrary location

Parameters

- **fig** (*int or None*) – The figure number. If None, a new figure number will be used
- **x0** (*float*) – the x-coordinate of the lower left corner (between 0 and 1)

- **y0** (*float*) – the y-coordinate of the lower left corner (between 0 and 1)
- **x1** (*float*) – the x-coordinate of the upper right corner (between 0 and 1)
- **y1** (*float*) – the y-coordinate of the upper right corner (between 0 and 1)

Methods

`create_axes`(x0, y0, x1, y1, **kwargs)	Create an axes for the given *fig*
`get_iter`()	Get the iterator over the axes
`resize_rectangle`(size)	resize the rectangle after changes of the widget size

static create_axes (*x0*, *y0*, *x1*, *y1*, ***kwargs*)
Create an axes for the given *fig*

Parameters

- **fig** (*int or None*) – The figure number. If None, a new figure number will be used
- **x0** (*float*) – the x-coordinate of the lower left corner (between 0 and 1)
- **y0** (*float*) – the y-coordinate of the lower left corner (between 0 and 1)
- **x1** (*float*) – the x-coordinate of the upper right corner (between 0 and 1)
- **y1** (*float*) – the y-coordinate of the upper right corner (between 0 and 1)
- ****kwargs** – Any other keyword argument for the `matplotlib.figure.Figure.add_axes()` method

get_iter ()
Get the iterator over the axes

resize_rectangle (*size*)
resize the rectangle after changes of the widget size

class psyplot_gui.plot_creator.**AxesCreatorCollection** (*key=None*, *func_kwargs={}*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QWidget`

Wrapper for a QToolBox that holds the different possibilities to select an axes

When the user finished, the `okpressed` symbol is emitted with an infinite iterator of strings. Possible widgets for the toolbox are determined by the `widgets` attribute

Parameters

- **key** (*str or None*) – if string, it must be one of the keys in the `widgets` attribute
- **func_kwargs** (*dict*) – a dictionary that is passed to the class constructor determined by the *key* parameter if *key* is not None
- ***args, **kwargs** – Determined by the QWidget class

Methods

`close`()	reimplemented to make sure that all widgets are closed when this one
`create_subplot`()	Method that is called whenn the ok button is pressed.

Attributes

okpressed(*args, **kwargs)	signal that is emitted when the 'Ok' pushbutton is pressed and the user
widgets	key, title and class fot the widget that is used to create an

close()
: reimplemented to make sure that all widgets are closed when this one is closed

create_subplot()
: Method that is called whenn the ok button is pressed.

  It emits the *okpressed* signal with the iterator of the current widget in the toolbox

okpressed(**args*, ***kwargs*)
: signal that is emitted when the 'Ok' pushbutton is pressed and the user finished the selection

widgets = [('subplot', 'Subplot in a grid', <class 'psyplot_gui.plot_creator.SubplotCr
: key, title and class fot the widget that is used to create an axes

class psyplot_gui.plot_creator.**AxesSelector**(**args*, ***kwargs*)
: Bases: `PyQt5.QtWidgets.QWidget`

  Widget to select an already created axes

  Click the button, select your axes and click the button again **Methods**

allow_axes_select()	Replace make all axes pickable
change_pickers(b)	Change the pickers of the axes instances
close()	Reimplemented to restore the pickers if the widget is closed
get_iter()	Get the iterator over the axes
get_picked_ax(event)	Function to be called when an axes is picked
inspect_axes(ax)	Inspect the given axes and get the right string for making a plot
restore_pickers()	Restore the original pickers of the existing axes instances
setVisible(b)	Reimplemented to restore the pickers if the widget is made invisible
unclick()	Restore the original pickers

allow_axes_select()
: Replace make all axes pickable

change_pickers(*b*)
: Change the pickers of the axes instances

  If the push button is clicked, we replace the existing pickers of the axes in order to select the plots. Otherwise we restore them

close()
: Reimplemented to restore the pickers if the widget is closed

get_iter()
: Get the iterator over the axes

get_picked_ax(*event*)
: Function to be called when an axes is picked

inspect_axes(*ax*)
Inspect the given axes and get the right string for making a plot with it

restore_pickers()
Restore the original pickers of the existing axes instances

setVisible(*b*)
Reimplemented to restore the pickers if the widget is made invisible

unclick()
Restore the original pickers

class psyplot_gui.plot_creator.**AxesViewer**(**args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QGraphicsView`

Widget to show a rectangle **Methods**

resizeEvent(self, QResizeEvent)	

Attributes

sizeChanged(*args, **kwargs)	

resizeEvent(*self*, *QResizeEvent*)

sizeChanged(**args*, ***kwargs*)

class psyplot_gui.plot_creator.**CoordComboBox**(*ds_func*, *dim*, *parent=None*)
Bases: `PyQt5.QtWidgets.QComboBox`

Combobox showing coordinate information of a dataset

This combobox loads its data from the current dataset and allows the popups to be left open. It also has a *leftclick* signal that is emitted when the popup is about to be closed because the user clicked on a value

Parameters

- **ds_func** (*function*) – The function that, when called without arguments, returns the xarray.Dataset to use
- **dim** (*str*) – The coordinate name for this combobox
- **parent** (*PyQt5.QtWidgets.QWidget*) – The parent widget

Attributes

close_popups	
leftclick(*args, **kwargs)	
use_coords	

Methods

eventFilter(obj, event)	Reimplemented to filter right-click events on the view()
handleItemPressed(index)	Function to be called when an item is pressed to make sure that
hidePopup()	Reimplemented to only close the popup when the `close_popup`

Continued on next page

Table 99 – continued from previous page

`hide_anyway`([index])	Function to hide the popup despite of the `_changed` attribute
`load_coord`()	Load the coordinate data from the dataset and fill the combobox with
`mouseDoubleClickEvent`(*args, **kwargs)	Reimplemented to fill the box with content from the dataset
`mousePressEvent`(*args, **kwargs)	Reimplemented to fill the box with content from the dataset
`right_click`(point)	Function that is called when an item is right_clicked

close_popups

eventFilter(*obj*, *event*)
Reimplemented to filter right-click events on the view()

handleItemPressed(*index*)
Function to be called when an item is pressed to make sure that we know whether anything changed before closing the popup

hidePopup()
Reimplemented to only close the popup when the `close_popup` attribute is True or it is clicked outside the window

hide_anyway(*index=None*)
Function to hide the popup despite of the `_changed` attribute

leftclick(**args*, ***kwargs*)

load_coord()
Load the coordinate data from the dataset and fill the combobox with it (if it is empty)

mouseDoubleClickEvent(**args*, ***kwargs*)
Reimplemented to fill the box with content from the dataset

mousePressEvent(**args*, ***kwargs*)
Reimplemented to fill the box with content from the dataset

right_click(*point*)
Function that is called when an item is right_clicked

use_coords

class `psyplot_gui.plot_creator.`**CoordsTable**(*get_func*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QTableWidget`

A table showing the coordinates of in a dataset via instances of `CoordComboBox`

Parameters

- **get_func** (*function*) – The function that, when called without arguments, returns the xarray.Dataset to use
- ****kwargs** (**args,*) – Determined by the `PyQt5.QtWidgets.QTableWidget` class

Attributes

`combo_boxes`	A list of `CoordComboBox` in this table

Methods

fill_from_ds([ds])	Clear the table and create new comboboxes
sizeHint()	Reimplemented to adjust the heigth based upon the header and the

combo_boxes
A list of *CoordComboBox* in this table

fill_from_ds (*ds=None*)
Clear the table and create new comboboxes

sizeHint ()
Reimplemented to adjust the heigth based upon the header and the first row

class psyplot_gui.plot_creator.**DragDropTable** (**args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QTableWidget`

Table that allows to exchange rows via drag and drop

This class was mainly taken from http://stackoverflow.com/questions/26227885/drag-and-drop-rows-within-qtablewidget **Methods**

dropEvent(self, QDropEvent)	
dropOn(event)	
droppingOnItself(event, index)	
moveRows(row[, remove])	Move all selected rows to the given *row*
position(pos, rect, index)	

dropEvent (*self*, *QDropEvent*)

dropOn (*event*)

droppingOnItself (*event*, *index*)

moveRows (*row*, *remove=False*)
Move all selected rows to the given *row*

position (*pos*, *rect*, *index*)

class psyplot_gui.plot_creator.**PlotCreator** (**args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QDialog`

Widget to extract data from a dataset and eventually create a plot **Attributes**

NO_PM_TT	Tooltip for not making a plot

Methods

add_new_ds(oname, ds[, fname])	
close(*args, **kwargs)	Reimplemented to make sure that the data sets are deleted
connect_combo_boxes()	
create_plots()	Method to be called when the *Create plot* button is pressed
fill_ds_combo(project)	fill the dataset combobox with datasets of the current main project
fill_fmt_tree(pm)	

Continued on next page

Table 104 – continued from previous page

fill_plot_method_combo()	Takes the names of the plotting methods in the current project
get_ds([i])	Get the dataset
insert_array([variables])	Inserts an array for the given variables (or the ones selected in
insert_array_from_combo(cb[, variables])	Insert new arrays into the dataset when the combobox is left-clicked
keyPressEvent(e)	Reimplemented to close the window when escape is hitted
open_data(*args, **kwargs)	Convenience method to create a sub project without a plotter
open_dataset([fnames])	Opens a file dialog and the dataset that has been inserted
reset_comboboxes()	Clear all comboboxes
set_ds(i)	Set the current dataset
set_pm(plot_method)	
setup_subplot()	Method to be emitted to setup one subplot at a specific location
setup_subplots()	Method to be emitted to setup the subplots for the selected arrays
show_pm_info()	Shows info on the current plotting method in the help explorer
switch2ds(ds)	Switch to the given dataset
toggle_close_popups()	Change the automatic closing of popups

NO_PM_TT = 'Choose a plot method (or choose none to only extract the data)'
Tooltip for not making a plot

add_new_ds(*oname*, *ds*, *fname=None*)

close(**args*, ***kwargs*)
Reimplemented to make sure that the data sets are deleted

connect_combo_boxes()

create_plots()
Method to be called when the *Create plot* button is pressed

This method reads the data from the `array_table` attribute and makes the plot (or extracts the data) based upon the `plot_method` attribute

fill_ds_combo(*project*)
fill the dataset combobox with datasets of the current main project

fill_fmt_tree(*pm*)

fill_plot_method_combo()
Takes the names of the plotting methods in the current project

get_ds(*i=None*)
Get the dataset

Parameters i (*int or None*) – If None, the dataset of the current index in the *ds_combo* is returned. Otherwise it specifies the locdation of the dictionary in the `ds_descs` attribute

Returns The requested dataset

Return type xarray.Dataset

insert_array(*variables=None*)
Inserts an array for the given variables (or the ones selected in the `variable_table` if *variables* is None)

insert_array_from_combo(*cb*, *variables=None*)
Insert new arrays into the dataset when the combobox is left-clicked

keyPressEvent(*e*)
Reimplemented to close the window when escape is hitted

open_data(**args*, ***kwargs*)
Convenience method to create a sub project without a plotter

This method is used when the `pm_combo` is empty

open_dataset(*fnames=None*, **args*, ***kwargs*)
Opens a file dialog and the dataset that has been inserted

reset_comboboxes()
Clear all comboboxes

set_ds(*i*)
Set the current dataset

set_pm(*plot_method*)

setup_subplot()
Method to be emitted to setup one subplot at a specific location for each of the selected arrays on separate (new) figures

setup_subplots()
Method to be emitted to setup the subplots for the selected arrays on new figures

show_pm_info()
Shows info on the current plotting method in the help explorer

switch2ds(*ds*)
Switch to the given dataset

Parameters **ds** (`xarray.Dataset`) – The dataset to use. It is assumed that this dataset is already in the dataset combobox

toggle_close_popups()
Change the automatic closing of popups

class psyplot_gui.plot_creator.**SubplotCreator**(*fig=None*, *rows=1*, *cols=1*, *num1=1*, *num2=None*, **args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QWidget`

Select a subplot to which will be created (if not already existing) when making the plot

Parameters

- **fig** (`int or None`) – The number of the figure
- **rows** (`int`) – The number of rows for the gridspec
- **cols** (`int`) – The number of columns for the gridspec
- **num1** (`int`) – The number of the upper left corner starting from 1
- **num2** (`int or None`) – The number of the lower right corner starting from 1. If None, *num1* is used

Methods

`create_subplot`([rows, cols, num1, num2])	Create a subplot for the given figure
`get_iter`()	Get the iterator over the axes
`set_num2_validator`(s)	Set the validator range for the num2 line edit
`set_selected`(num1, num2)	Update the selection in the table based upon *num1* and *num2*
`set_selected_from_num1`(s)	Update the selection of the table after changes of
`set_selected_from_num2`(s)	Update the selection of the table after changes of `num2_edit`
`setup_table`()	Set up the table based upon the number of rows and columns in the
`update_num_edit`()	Update the `num1_edit` and `num2_edit` after the

static create_subplot(*rows=1*, *cols=1*, *num1=1*, *num2=None*, ***kwargs*)
Create a subplot for the given figure

Parameters

- **fig** (`matplotlib.figure.Figure` or int) – If integer, the `matplotlib.pyplot.figure()` function is used
- **rows** (*int*) – Number of rows for the gridspec
- **cols** (*int*) – Number of columns for the gridspec
- **num1** (*int*) – The subplot number of the upper left corner in the grid (starting from 1!)
- **num2** (*None or int*) – The subplot number of the lower left corner in the grid (starting from 1!). If None, *num1* will be used
- ****kwargs** – Any other keyword argument for the `matplotlib.figure.Figure.add_subplot()` method

Returns The new created subplot

Return type mpl.axes.Subplot

get_iter()
Get the iterator over the axes

set_num2_validator(*s*)
Set the validator range for the num2 line edit

set_selected(*num1*, *num2*)
Update the selection in the table based upon *num1* and *num2*

set_selected_from_num1(*s*)
Update the selection of the table after changes of `num1_edit`

set_selected_from_num2(*s*)
Update the selection of the table after changes of `num2_edit`

setup_table()
Set up the table based upon the number of rows and columns in the rows and cols line edit

update_num_edit()
Update the `num1_edit` and `num2_edit` after the selection of the table changed

class psyplot_gui.plot_creator.**VariableItemDelegate**
Bases: `PyQt5.QtWidgets.QStyledItemDelegate`

Delegate alowing only the variables in the parents dataset.

The parent must hold a *get_ds* method that returns a dataset when called **Methods**

createEditor(self, QWidget, ...)	

createEditor (*self*, *QWidget*, *QStyleOptionViewItem*, *QModelIndex*) → QWidget

class psyplot_gui.plot_creator.**VariablesTable** (*get_func*, *columns=['long_name', 'dims', 'shape']*, **args*, ***kwargs*)

Bases: `PyQt5.QtWidgets.QTableWidget`

Table to display the variables of a dataset

Parameters

- **get_func** (*function*) – The function that, when called without arguments, returns the xarray.Dataset to use
- **columns** (*list of str*) – The attribute that will be used as columns for the variables

Methods

fill_from_ds([ds])	Clear the table and insert items from the given *dataset*
set_columns([columns])	

Attributes

selected_variables	The currently selected variables
variables	The variables in the dataset

fill_from_ds (*ds=None*)
Clear the table and insert items from the given *dataset*

selected_variables
The currently selected variables

set_columns (*columns=None*)

variables
The variables in the dataset

psyplot_gui.preferences module

Preferences widget for psyplot_gui

This module defines the `Preferences` widget that creates an interface to the rcParams of psyplot and psyplot_gui

Classes

ConfigPage	An abstract base class for configuration pages
GuiRcParamsWidget(*args, **kwargs)	The config page for the *psyplot_gui.config.rcsetup.rcParams*
Prefences([main])	Preferences dialog
PsyRcParamsWidget(*args, **kwargs)	The config page for the `psyplot.config.rcsetup.rcParams`

Continued on next page

Table 109 – continued from previous page

RcParamsTree(rcParams, validators, . . .)	A QTreeWidget that can be used to display a RcParams instance
RcParamsWidget(*args, **kwargs)	A configuration page for RcParams instances

class psyplot_gui.preferences.**ConfigPage**

Bases: `object`

An abstract base class for configuration pages **Methods**

apply_changes()	Apply the planned changes
initialize()	Initialize the page

Attributes

auto_updates	`bool` that is True, if the changes in this ConfigPage are set
changed	Check whether the preferences will change
icon	The icon of the page
is_valid	Check whether the page is valid
propose_changes(*args, **kwargs)	A signal that is emitted if changes are propsed.
title	The title for the config page
validChanged(*args, **kwargs)	A signal that shall be emitted if the validation state changes

apply_changes()

Apply the planned changes

auto_updates = False

`bool` that is True, if the changes in this ConfigPage are set immediately

changed

Check whether the preferences will change

icon = None

The icon of the page

initialize()

Initialize the page

is_valid

Check whether the page is valid

propose_changes(**args*, ***kwargs*)

A signal that is emitted if changes are propsed. The signal should be emitted with the instance of the page itself

title = None

The title for the config page

validChanged(**args*, ***kwargs*)

A signal that shall be emitted if the validation state changes

class psyplot_gui.preferences.**GuiRcParamsWidget**(**args*, ***kwargs*)

Bases: *psyplot_gui.preferences.RcParamsWidget*

The config page for the *psyplot_gui.config.rcsetup.rcParams* **Attributes**

default_path	str(object=’‘) -> str
rc	RcParams for the psyplot-gui package.
title	str(object=’‘) -> str

default_path = '/home/docs/.config/psyplot/psyplotguirc.yml'

rc = {'backend': 'psyplot', 'console.auto_set_mp': True, 'console.auto_set_sp': Tru

title = 'GUI defaults'

class psyplot_gui.preferences.**Prefences**(*main=None*)

Bases: `PyQt5.QtWidgets.QDialog`

Preferences dialog **Methods**

accept()	Reimplement Qt method
add_page(widget)	Add a new page to the preferences dialog
apply_clicked()	
check_changes(configpage)	Enable the apply button if there are changes to the settings
current_page_changed(index)	
get_page([index])	Return page widget
load_plugin_pages()	Load the rcParams for the plugins in separate pages
set_current_index(index)	Set current page index

Attributes

bt_apply
pages

accept()

Reimplement Qt method

add_page(*widget*)

Add a new page to the preferences dialog

Parameters widget (`ConfigPage`) – The page to add

apply_clicked()

bt_apply

check_changes(*configpage*)

Enable the apply button if there are changes to the settings

current_page_changed(*index*)

get_page(*index=None*)

Return page widget

load_plugin_pages()

Load the rcParams for the plugins in separate pages

pages

set_current_index(*index*)

Set current page index

class psyplot_gui.preferences.**PsyRcParamsWidget**(**args, **kwargs*)

Bases: *psyplot_gui.preferences.RcParamsWidget*

The config page for the `psyplot.config.rcsetup.rcParams` **Attributes**

default_path	str(object='') -> str
rc	A dictionary object including validation
title	str(object='') -> str

default_path = '/home/docs/.config/psyplot/psyplotrc.yml'

rc = {'auto_draw': True, 'auto_show': False, 'colors.cmaps': {}, 'datapath': None,

title = 'psyplot defaults'

class psyplot_gui.preferences.**RcParamsTree**(*rcParams, validators, descriptions, *args, **kwargs*)

Bases: `PyQt5.QtWidgets.QTreeWidget`

A QTreeWidget that can be used to display a RcParams instance

This widget is populated by a `psyplot.config.rcsetup.RcParams` instance and displays whether the values are valid or not

Parameters

- **rcParams** (*dict*) – The dictionary that contains the rcParams
- **validators** (*dict*) – A mapping from the *rcParams* key to the validation function for the corresponding value
- **descriptions** (*dict*) – A mapping from the *rcParams* key to it's description

Methods

apply_changes()	Update the `rc` with the proposed changes
changed_rc([use_items])	Iterate over the changed rcParams
initialize()	Fill the items of the `rc` into the tree
open_menu(position)	Open a menu to expand and collapse all items in the tree
select_changes()	Select all the items that changed comparing to the current rcParams
selected_rc([use_items])	Iterate over the selected rcParams
set_icon_func(i, item, validator)	Create a function to change the icon of one topLevelItem
set_valid(i, b)	Set the validation status

Attributes

is_valid	True if all the proposed values in this tree are valid
propose_changes(*args, **kwargs)	A signal that is emitted if changes are propsed.
rc	The `RcParams` to display
top_level_items	An iterator over the topLevelItems in this tree
valid	list of `bool`. A boolean for each rcParams key that states
validChanged(*args, **kwargs)	A signal that shall be emitted if the validation state changes
value_col	int(x=0) -> integer

See also:

`psyplot.config.rcsetup.RcParams`, `psyplot.config.rcsetup.RcParams.validate`, `psyplot.config.rcsetup.RcParams.descriptions`

`apply_changes`()
Update the `rc` with the proposed changes

`changed_rc`(*use_items=False*)
Iterate over the changed rcParams

Parameters **`use_items`** (`bool`) – If True, the topLevelItems are used instead of the keys

Yields

- *QTreeWidgetItem or str* – The item identifier
- *object* – The proposed value

`initialize`()
Fill the items of the `rc` into the tree

`is_valid`
True if all the proposed values in this tree are valid

`open_menu`(*position*)
Open a menu to expand and collapse all items in the tree

Parameters **`position`** (`QPosition`) – The position where to open the menu

`propose_changes`(**args*, ***kwargs*)
A signal that is emitted if changes are propsed. It is either emitted with the parent of this instance (if this is not None) or with the instance itself

`rc = None`
The `RcParams` to display

`select_changes`()
Select all the items that changed comparing to the current rcParams

`selected_rc`(*use_items=False*)
Iterate over the selected rcParams

Parameters **`use_items`** (`bool`) – If True, the topLevelItems are used instead of the keys

Yields

- *QTreeWidgetItem or str* – The item identifier
- *object* – The proposed value

`set_icon_func`(*i*, *item*, *validator*)
Create a function to change the icon of one topLevelItem

This method creates a function that can be called when the value of an item changes to display it's valid state. The returned function changes the icon of the given topLevelItem depending on whether the proposed changes are valid or not and it modifies the `valid` attribute accordingly

Parameters

- **`i`** (`int`) – The index of the topLevelItem
- **`item`** (`QTreeWidgetItem`) – The topLevelItem
- **`validator`** (`func`) – The validation function

Returns The function that can be called to set the correct icon

Return type function

set_valid(*i*, *b*)
Set the validation status

If the validation status changed compared to the old one, the *validChanged* signal is emitted

Parameters

- **i** (*int*) – The index of the topLevelItem
- **b** (*bool*) – The valid state of the item

top_level_items
An iterator over the topLevelItems in this tree

valid = []
list of bool. A boolean for each rcParams key that states whether the proposed value is valid or not

validChanged(**args*, ***kwargs*)
A signal that shall be emitted if the validation state changes

value_col = 2

class psyplot_gui.preferences.**RcParamsWidget**(**args*, ***kwargs*)
Bases: *psyplot_gui.preferences.ConfigPage*, PyQt5.QtWidgets.QWidget

A configuration page for RcParams instances

This page displays the psyplot.config.rcsetup.RcParams instance in the *rc* attribute and let's the user modify it.

Notes

Methods

apply_changes()	Apply the changes in the config page
initialize([rcParams, validators, descriptions])	Initialize the config page
save_settings_action([update, target])	Create an action to save the selected settings in the tree

Attributes

changed	True if any changes are proposed by this config page
icon	The icon of this instance in the Preferences dialog
is_valid	True if all the settings are valid
propose_changes	A signal that is emitted if the user changes the values in the
rc	the rcParams to use (must be implemented by subclasses)
tree	the *RcParamsTree* that is used to display the rcParams
validChanged	A signal that is emitted if the user changes the valid state of this

After the initialization, you have to call the *initialize()* method

apply_changes()

Apply the changes in the config page

changed
True if any changes are proposed by this config page

icon
The icon of this instance in the `Preferences` dialog

initialize(*rcParams=None*, *validators=None*, *descriptions=None*)
Initialize the config page

Parameters

- **rcParams** (`dict`) – The rcParams to use. If None, the `rc` attribute of this instance is used
- **validators** (`dict`) – A mapping from the *rcParams* key to the corresponding validation function for the value. If None, the `validate` attribute of the `rc` attribute is used
- **descriptions** (`dict`) – A mapping from the *rcParams* key to it's description. If None, the `descriptions` attribute of the `rc` attribute is used

is_valid
True if all the settings are valid

propose_changes
A signal that is emitted if the user changes the values in the rcParams

rc = None
the rcParams to use (must be implemented by subclasses)

save_settings_action(*update=False*, *target=None*)
Create an action to save the selected settings in the `tree`

Parameters **update** (`bool`) – If True, it is expected that the file already exists and it will be updated. Otherwise, existing files will be overwritten

tree = None
the `RcParamsTree` that is used to display the rcParams

validChanged
A signal that is emitted if the user changes the valid state of this page

psyplot_gui.version module

1.7 Changelog

1.7.1 v1.1.0

This release mainly adds the possibility to create plugins into the psyplot-gui and it adds a new framework to allow the formatoptions to provide a custom interface to the formatoptions widget.

Added

- Added layout windows menu and default layout
- Added `script` and `command` command line arguments
- The `pwd` command line arguments now changes the working directory of the running GUI

- Added callbacks to the `MainWindow` class. This framework can be used on a low level to interact with the current GUI.
- The DataFrameEditor. A widget to display dataframes
- The implementation of the `psyplot.plotter.Formatoption.get_fmt_widget` method. Formatoptions now can add a custom widget to the formatoptions widget

1.7.2 v1.0.1

Added

- added changelog

Changed

- fixed bug that prevented startup on Windows

1.8 ToDos

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index