psy-simple Documentation

Release 1.1.0

Philipp Sommer

Apr 09, 2018

Contents

Welcome to the psyplot plugin for simple visualization. This package targets simple visualization like line plots, 2D plots, bar plots, density plots, etc. It provides the basics for all the more advanced and specialized plugins like the psy-maps or psy-reg plugin.

See the *psyplot plot methods* and *Example Gallery* for more information.

CHAPTER 1

Documentation

1.1 Installation

1.1.1 How to install

Installation using conda

We highly recommend to use conda for installing psy-simple. After downloading the installer from anaconda, you can install psy-simple simply via:

```
$ conda install -c conda-forge psy-simple
```

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 psy-simple
```

1.1.2 Running the tests

First, clone out the github repository. First you have to

- either checkout the reference figures via:

  ```
  $ git submodule update --init `python tests/get_ref_dir.py`
  ```

- or create the reference figures via:

  ```
  $ python setup.py test -a "--ref"
  ```

After that, you can run:

```
$ python setup.py test
```

or after having install pytest:

```
$ py.test
```

1.2 psyplot plot methods

This plugin defines the following new plot methods for the `psyplot.project.ProjectPlotter` class. They can, for example, be accessed through

```
In [1]: import psyplot.project as psy

In [2]: psy.plot.lineplot
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-2-dfdce4ecf5a0> in <module>()
----> 1 psy.plot.lineplot

AttributeError: 'ProjectPlotter' object has no attribute 'lineplot'
```

psyplot.project.plot.lineplot
psyplot.project.plot.vector
psyplot.project.plot.violinplot
psyplot.project.plot.plot2d
psyplot.project.plot.combined
psyplot.project.plot.density
psyplot.project.plot.barplot
psyplot.project.plot.fldmean

1.2.1 psyplot.project.plot.lineplot

plot.**lineplot**(**args*, ***kwargs*)

Make a line plot of one-dimensional data

This plotting method adds data arrays and plots them via *psy_simple.plotters.LinePlotter* plotters

To plot data from a netCDF file type:

```
>>> psy.plot.lineplot(filename, name=['my_variable'], ...)
```

Possible formatoptions are

axiscolor	*color*	*coord*	*error*
erroralpha	*figtitle*	*figtitleprops*	*figtitlesize*
figtitleweight	*grid*	*labelprops*	*labelsize*
labelweight	*legend*	*legendlabels*	*linewidth*
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*

Examples

To explore the formatoptions and their documentations, use the `keys`, `summaries` and `docs` methods. For example:

```
>>> import psyplot.project as psy

# show the keys corresponding to a group or multiple
# formatopions
>>> psy.plot.lineplot.keys('labels')

# show the summaries of a group of formatoptions or of a
# formatoption
>>> psy.plot.lineplot.summaries('title')

# show the full documentation
>>> psy.plot.lineplot.docs('plot')

# or access the documentation via the attribute
>>> psy.plot.lineplot.plot
```

1.2.2 psyplot.project.plot.vector

`plot.`**`vector`**`(*args, **kwargs)`

Make a simple plot of a 2D vector field

This plotting method adds data arrays and plots them via *psy_simple.plotters.SimpleVectorPlotter* plotters

To plot data from a netCDF file type:

```
>>> psy.plot.vector(filename, name=[['u_var', 'v_var']], ...)
```

Possible formatoptions are

arrowsize	*arrowstyle*	*axiscolor*	*bounds*
cbar	*cbarspacing*	*clabel*	*clabelprops*
clabelsize	*clabelweight*	*cmap*	*color*
cticklabels	*ctickprops*	*cticks*	*cticksize*
ctickweight	*datagrid*	*density*	*extend*
figtitle	*figtitleprops*	*figtitlesize*	*figtitleweight*
grid	*labelprops*	*labelsize*	*labelweight*
linewidth	*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*	

Examples

To explore the formatoptions and their documentations, use the `keys`, `summaries` and `docs` methods. For example:

```
>>> import psyplot.project as psy

# show the keys corresponding to a group or multiple
# formatopions
>>> psy.plot.vector.keys('labels')

# show the summaries of a group of formatoptions or of a
# formatoption
>>> psy.plot.vector.summaries('title')

# show the full documentation
>>> psy.plot.vector.docs('plot')

# or access the documentation via the attribute
>>> psy.plot.vector.plot
```

1.2.3 psyplot.project.plot.violinplot

`plot.violinplot`(**args*, ***kwargs*)

Make a violin plot of your data

This plotting method adds data arrays and plots them via *psy_simple.plotters.ViolinPlotter* plotters

To plot data from a netCDF file type:

```
>>> psy.plot.violinplot(filename, name=['my_variable'], ...)
```

Possible formatoptions are

axiscolor	*color*	*figtitle*	*figtitleprops*
figtitlesize	*figtitleweight*	*grid*	*labelprops*
labelsize	*labelweight*	*legend*	*legendlabels*
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*		

Examples

To explore the formatoptions and their documentations, use the `keys`, `summaries` and `docs` methods. For example:

```
>>> import psyplot.project as psy

# show the keys corresponding to a group or multiple
# formatopions
>>> psy.plot.violinplot.keys('labels')

# show the summaries of a group of formatoptions or of a
# formatoption
>>> psy.plot.violinplot.summaries('title')

# show the full documentation
>>> psy.plot.violinplot.docs('plot')

# or access the documentation via the attribute
>>> psy.plot.violinplot.plot
```

1.2.4 psyplot.project.plot.plot2d

`plot.`**`plot2d`**`(`**args*, ***kwargs*`)`

Make a simple plot of a 2D scalar field

This plotting method adds data arrays and plots them via *psy_simple.plotters.Simple2DPlotter* plotters

To plot data from a netCDF file type:

```
>>> psy.plot.plot2d(filename, name=['my_variable'], ...)
```

Possible formatoptions are

axiscolor	bounds	cbar	cbarspacing
clabel	clabelprops	clabelsize	clabelweight
cmap	cticklabels	ctickprops	cticks
cticksize	ctickweight	datagrid	extend
figtitle	figtitleprops	figtitlesize	figtitleweight
grid	interp_bounds	labelprops	labelsize
labelweight	levels	maskbetween	maskgeq
maskgreater	maskleq	maskless	miss_color
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			

Examples

To explore the formatoptions and their documentations, use the keys, summaries and docs methods. For example:

```
>>> import psyplot.project as psy

# show the keys corresponding to a group or multiple
# formatopions
>>> psy.plot.plot2d.keys('labels')

# show the summaries of a group of formatoptions or of a
# formatoption
>>> psy.plot.plot2d.summaries('title')

# show the full documentation
>>> psy.plot.plot2d.docs('plot')

# or access the documentation via the attribute
>>> psy.plot.plot2d.plot
```

1.2.5 psyplot.project.plot.combined

plot.**combined**(**args*, ***kwargs*)

Plot a 2D scalar field with an overlying vector field

This plotting method adds data arrays and plots them via psy_simple.plotters.CombinedSimplePlotter plotters

To plot data from a netCDF file type:

```
>>> psy.plot.combined(filename, name=[['my_variable', ['u_var', 'v_var']]], ...)
```

Possible formatoptions are

arrowsize	arrowstyle	axiscolor	bounds
cbar	cbarspacing	clabel	clabelprops
clabelsize	clabelweight	cmap	color
cticklabels	ctickprops	cticks	cticksize
ctickweight	datagrid	density	extend
figtitle	figtitleprops	figtitlesize	figtitleweight
grid	interp_bounds	labelprops	labelsize
labelweight	levels	linewidth	maskbetween
maskgeq	maskgreater	maskleq	maskless
miss_color	plot	post	post_timing
sym_lims	text	ticksize	tickweight
tight	title	titleprops	titlesize
titleweight	transpose	vbounds	vcbar
vcbarspacing	vclabel	vclabelprops	vclabelsize
vclabelweight	vcmap	vcticklabels	vctickprops
vcticks	vcticksize	vctickweight	vplot
xlabel	xlim	xrotation	xticklabels
xtickprops	xticks	ylabel	ylim
yrotation	yticklabels	ytickprops	yticks

Examples

To explore the formatoptions and their documentations, use the `keys`, `summaries` and `docs` methods. For example:

```
>>> import psyplot.project as psy

# show the keys corresponding to a group or multiple
# formatopions
>>> psy.plot.combined.keys('labels')

# show the summaries of a group of formatoptions or of a
# formatoption
>>> psy.plot.combined.summaries('title')

# show the full documentation
>>> psy.plot.combined.docs('plot')

# or access the documentation via the attribute
>>> psy.plot.combined.plot
```

1.2.6 psyplot.project.plot.density

plot.**density**(**args*, ***kwargs*)

Make a density plot of point data

This plotting method adds data arrays and plots them via *psy_simple.plotters.DensityPlotter* plotters

To plot data from a netCDF file type:

```
>>> psy.plot.density(filename, name=['my_variable'], ...)
```

Possible formatoptions are

axiscolor	*bins*	*bounds*	*cbar*
cbarspacing	*clabel*	*clabelprops*	*clabelsize*
clabelweight	*cmap*	*coord*	*cticklabels*
ctickprops	*cticks*	*cticksize*	*ctickweight*
datagrid	*density*	*extend*	*figtitle*
figtitleprops	*figtitlesize*	*figtitleweight*	*grid*
interp_bounds	*labelprops*	*labelsize*	*labelweight*
levels	*maskbetween*	*maskgeq*	*maskgreater*
maskleq	*maskless*	*miss_color*	*normed*
plot	*post*	*post_timing*	*precision*
sym_lims	*text*	*ticksize*	*tickweight*
tight	*title*	*titleprops*	*titlesize*
titleweight	*transpose*	*xlabel*	*xlim*
xrange	*xrotation*	*xticklabels*	*xtickprops*
xticks	*ylabel*	*ylim*	*yrange*
yrotation	*yticklabels*	*ytickprops*	*yticks*

Examples

To explore the formatoptions and their documentations, use the `keys`, `summaries` and `docs` methods. For example:

```
>>> import psyplot.project as psy

# show the keys corresponding to a group or multiple
# formatopions
>>> psy.plot.density.keys('labels')

# show the summaries of a group of formatoptions or of a
# formatoption
>>> psy.plot.density.summaries('title')

# show the full documentation
>>> psy.plot.density.docs('plot')

# or access the documentation via the attribute
>>> psy.plot.density.plot
```

1.2.7 psyplot.project.plot.barplot

`plot.`**`barplot`**(**args*, ***kwargs*)

Make a bar plot of one-dimensional data

This plotting method adds data arrays and plots them via *psy_simple.plotters.BarPlotter* plotters

To plot data from a netCDF file type:

```
>>> psy.plot.barplot(filename, name=['my_variable'], ...)
```

Possible formatoptions are

alpha	*axiscolor*	*color*	*coord*
figtitle	*figtitleprops*	*figtitlesize*	*figtitleweight*
grid	*labelprops*	*labelsize*	*labelweight*
legend	*legendlabels*	*maskbetween*	*maskgeq*
maskgreater	*maskleq*	*maskless*	*plot*
post	*post_timing*	*sym_lims*	*text*
ticksize	*tickweight*	*tight*	*title*
titleprops	*titlesize*	*titleweight*	*transpose*
widths	*xlabel*	*xlim*	*xrotation*
xticklabels	*xtickprops*	*xticks*	*ylabel*
ylim	*yrotation*	*yticklabels*	*ytickprops*
yticks			

Examples

To explore the formatoptions and their documentations, use the `keys`, `summaries` and `docs` methods. For example:

```
>>> import psyplot.project as psy

# show the keys corresponding to a group or multiple
# formatopions
>>> psy.plot.barplot.keys('labels')

# show the summaries of a group of formatoptions or of a
# formatoption
>>> psy.plot.barplot.summaries('title')

# show the full documentation
>>> psy.plot.barplot.docs('plot')

# or access the documentation via the attribute
>>> psy.plot.barplot.plot
```

1.2.8 psyplot.project.plot.fldmean

plot.**fldmean**(**args*, ***kwargs*)

Calculate and plot the mean over x- and y-dimensions

This plotting method adds data arrays and plots them via *psy_simple.plotters.FldmeanPlotter* plotters

To plot data from a netCDF file type:

```
>>> psy.plot.fldmean(filename, name=['my_variable'], ...)
```

Possible formatoptions are

axiscolor	*color*	*coord*	*err_calc*
error	*erroralpha*	*figtitle*	*figtitleprops*
figtitlesize	*figtitleweight*	*grid*	*labelprops*
labelsize	*labelweight*	*legend*	*legendlabels*
linewidth	*marker*	*markersize*	*maskbetween*
maskgeq	*maskgreater*	*maskleq*	*maskless*
mean	*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*		

Examples

To explore the formatoptions and their documentations, use the `keys`, `summaries` and `docs` methods. For example:

```
>>> import psyplot.project as psy

# show the keys corresponding to a group or multiple
# formatopions
>>> psy.plot.fldmean.keys('labels')

# show the summaries of a group of formatoptions or of a
# formatoption
>>> psy.plot.fldmean.summaries('title')

# show the full documentation
>>> psy.plot.fldmean.docs('plot')

# or access the documentation via the attribute
>>> psy.plot.fldmean.plot
```

1.3 Example Gallery

The psy-simple module provides you some plotters that are designed for very simple, non-projected visualization. Those range from simple line plots, over violin plots to 2-dimensional plots.

1.3.1 Line plot demo

This example shows you how to make a line plot using the `psyplot.project.ProjectPlotter.lineplot` method.

```
import psyplot.project as psy
```

```
axes = iter(psy.multiple_subplots(2, 2, n=3))
for var in ['t2m', 'u', 'v']:
```

(continues on next page)

(continued from previous page)

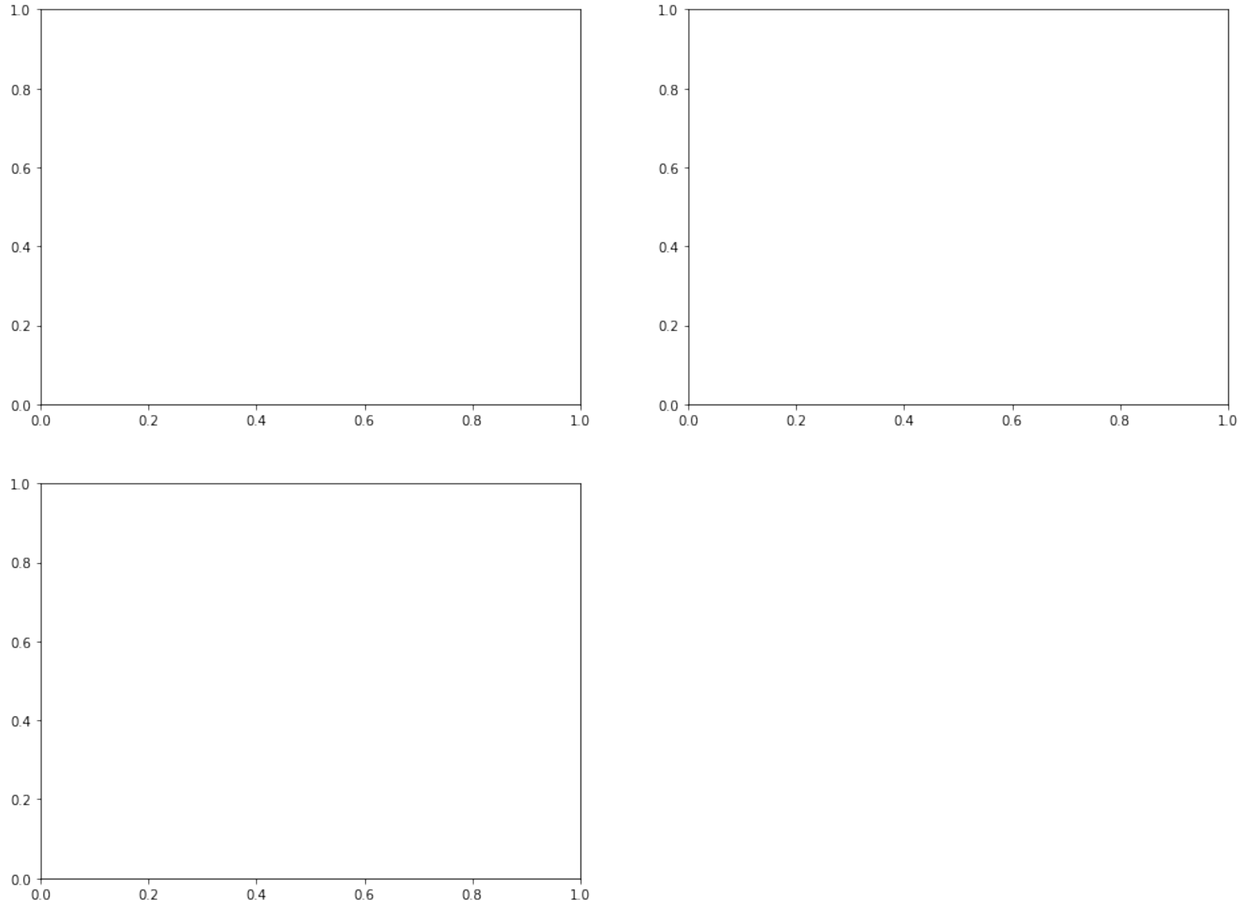
```
    psy.plot.lineplot(
        'demo.nc',  # netCDF file storing the data
        name=var, # one plot for each variable
        t=range(5),  # one violin plot for each time step
        z=0, x=0,      # choose latitude and longitude as dimensions
        ylabel="{desc}",  # use the longname and units on the y-axis
        ax=next(axes),
        color='coolwarm', legend=False
    )
lines = psy.gcp(True)
lines.show()
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-3-37b41cb32c58> in <module>()
      8         ylabel="{desc}",  # use the longname and units on the y-axis
      9         ax=next(axes),
---> 10         color='coolwarm', legend=False
     11     )
     12 lines = psy.gcp(True)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in __call__(self, *args, **kwargs)
   1398         """
   1399         return self.project._add_data(
-> 1400             self.plotter_cls, *args, **dict(chain(
   1401                 [('prefer_list', self._prefer_list),
   1402                  ('default_slice', self._default_slice)],


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in plotter_cls(self)
   1345         if ret is None:
   1346             self.project.logger.debug('importing %s', self.module)
-> 1347             mod = import_module(self.module)
   1348             plotter = self.plotter_name
   1349             if plotter not in vars(mod):


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/__init__.py in import_module(name, package)
    124                 break
    125             level += 1
--> 126     return _bootstrap._gcd_import(name[level:], package, level)
    127
    128


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _gcd_import(name, package, level)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load(name, import_)
```

(continues on next page)

(continued from previous page)

```
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load_unlocked(name, import_)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _load_unlocked(spec)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap_external.py in exec_module(self, module)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _call_with_frames_removed(f, *args, **kwds)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in <module>()
   5512
   5513
-> 5514 class FldmeanPlotter(LinePlotter):
   5515
   5516     _rcparams_string = ["plotter.fldmean."]


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in FldmeanPlotter()
   5523     # We reimplement the masking formatoption to make sure, that they are
   5524     # called after the mean calculation
-> 5525     maskgeq = MaskGeq('maskgeq', additional_children=['err_calc'])
   5526     maskleq = MaskLeq('maskleq', additional_children=['err_calc'])
   5527     maskgreater = MaskGreater('maskgreater', additional_children=['err_calc'])


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/plotter.py in __init__(self, key, plotter, index_in_list, **kwargs)
    429             raise TypeError(
    430                 '%s.__init__() got an unexpected keyword argument %r' % (
--> 431                     self.__class__.__name__, key))
    432         # set up child mapping
    433         self._child_mapping.update(kwargs)


TypeError: MaskGeq.__init__() got an unexpected keyword argument 'additional_children'
```




```
psy.close('all')
```

1.3.2 Bar plot demo

This example shows you how to make a bar plot using the `psyplot.project.ProjectPlotter.barplot` method.

```
import psyplot.project as psy
```

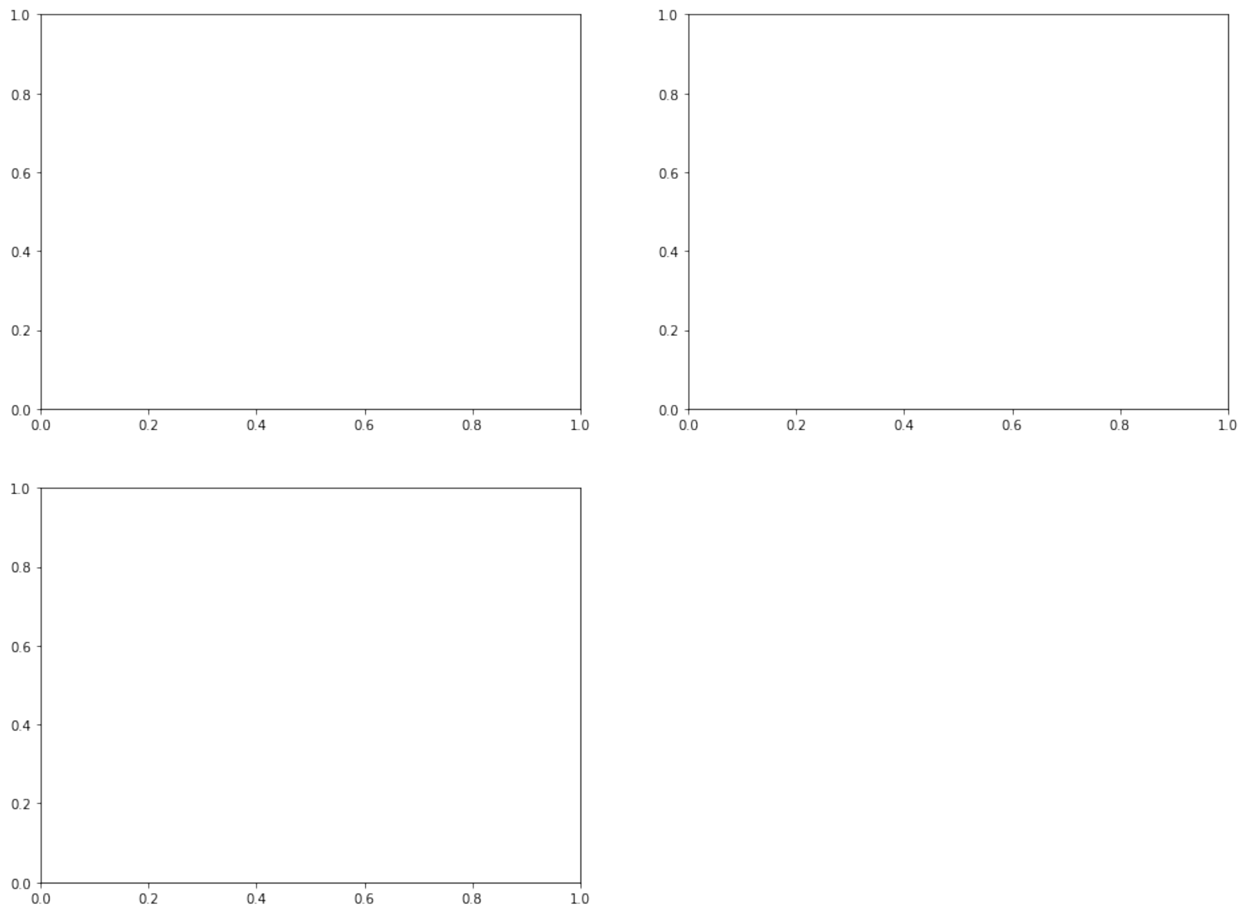
```
axes = iter(psy.multiple_subplots(2, 2, n=3))
for var in ['t2m', 'u', 'v']:
    psy.plot.barplot(
        'demo.nc',  # netCDF file storing the data
        name=var, # one plot for each variable
        y=[0, 1],  # two bars in total
        z=0, x=0,       # choose latitude and longitude as dimensions
        ylabel="{desc}",  # use the longname and units on the y-axis
        ax=next(axes),
        color='coolwarm', xticklabels='%B %Y',
        legendlabels='latitude %(y)1.2f $^\circ$N', legend='upper left',
        title='equally spaced'
    )
bars = psy.gcp(True)
bars.show()
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-3-70e7897760de> in <module>()
     10         color='coolwarm', xticklabels='%B %Y',
     11         legendlabels='latitude %(y)1.2f $^\circ$N', legend='upper left',
---> 12         title='equally spaced'
     13     )
     14 bars = psy.gcp(True)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in __call__(self, *args, **kwargs)
   1398         """
   1399         return self.project._add_data(
-> 1400             self.plotter_cls, *args, **dict(chain(
   1401                 [('prefer_list', self._prefer_list),
   1402                  ('default_slice', self._default_slice)],


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in plotter_cls(self)
   1345         if ret is None:
   1346             self.project.logger.debug('importing %s', self.module)
-> 1347             mod = import_module(self.module)
   1348             plotter = self.plotter_name
   1349             if plotter not in vars(mod):


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/__init__.py in import_module(name, package)
    124                 break
    125             level += 1
--> 126     return _bootstrap._gcd_import(name[level:], package, level)
    127
    128


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _gcd_import(name, package, level)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load(name, import_)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load_unlocked(name, import_)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _load_unlocked(spec)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap_external.py in exec_module(self, module)
```

(continues on next page)

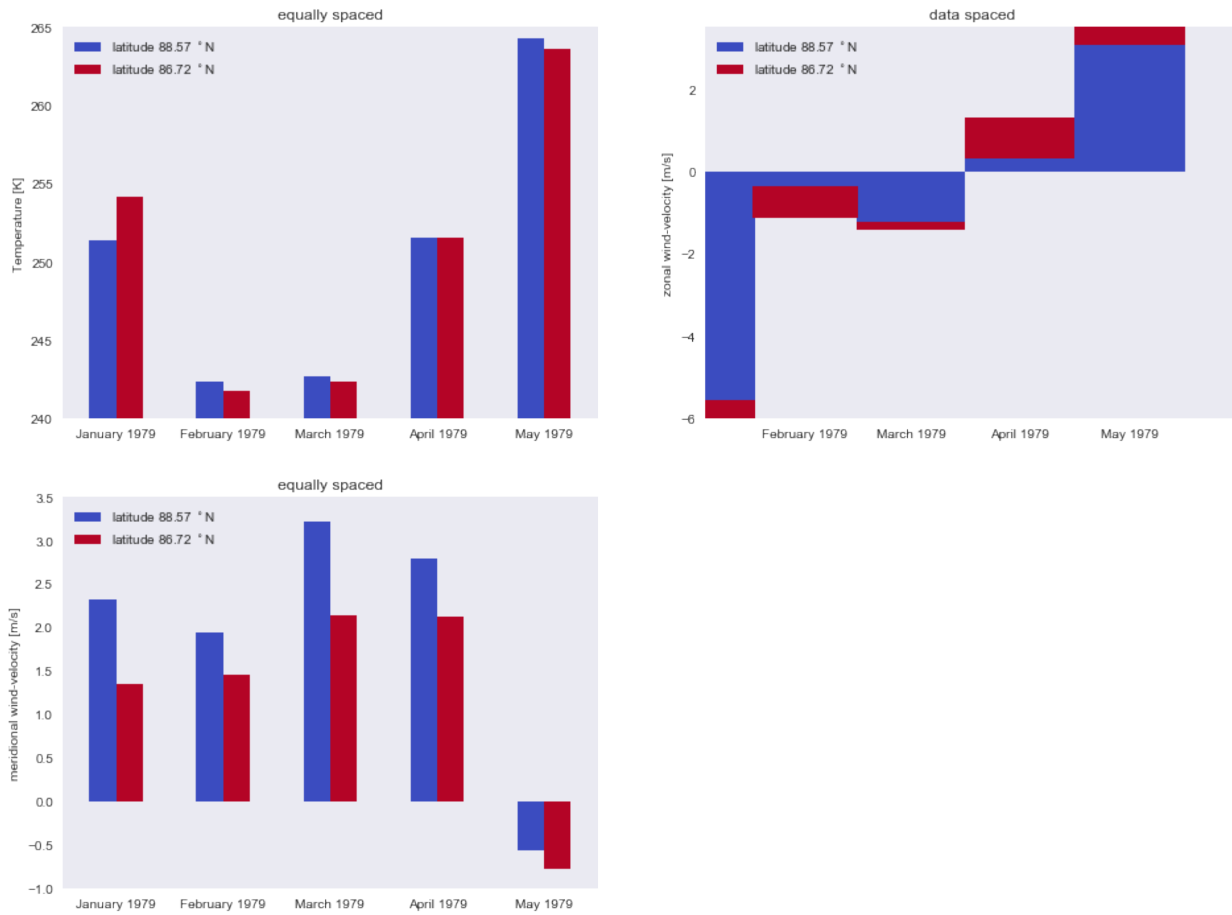
(continued from previous page)

```
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _call_with_frames_removed(f, *args, **kwds)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in <module>()
   5512
   5513
-> 5514 class FldmeanPlotter(LinePlotter):
   5515
   5516     _rcparams_string = ["plotter.fldmean."]


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in FldmeanPlotter()
   5523     # We reimplement the masking formatoption to make sure, that they are
   5524     # called after the mean calculation
-> 5525     maskgeq = MaskGeq('maskgeq', additional_children=['err_calc'])
   5526     maskleq = MaskLeq('maskleq', additional_children=['err_calc'])
   5527     maskgreater = MaskGreater('maskgreater', additional_children=['err_calc'])


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/plotter.py in __init__(self, key, plotter, index_in_list, **kwargs)
    429             raise TypeError(
    430                 '%s.__init__() got an unexpected keyword argument %r' % (
--> 431                     self.__class__.__name__, key))
    432         # set up child mapping
    433         self._child_mapping.update(kwargs)


TypeError: MaskGeq.__init__() got an unexpected keyword argument 'additional_children'
```



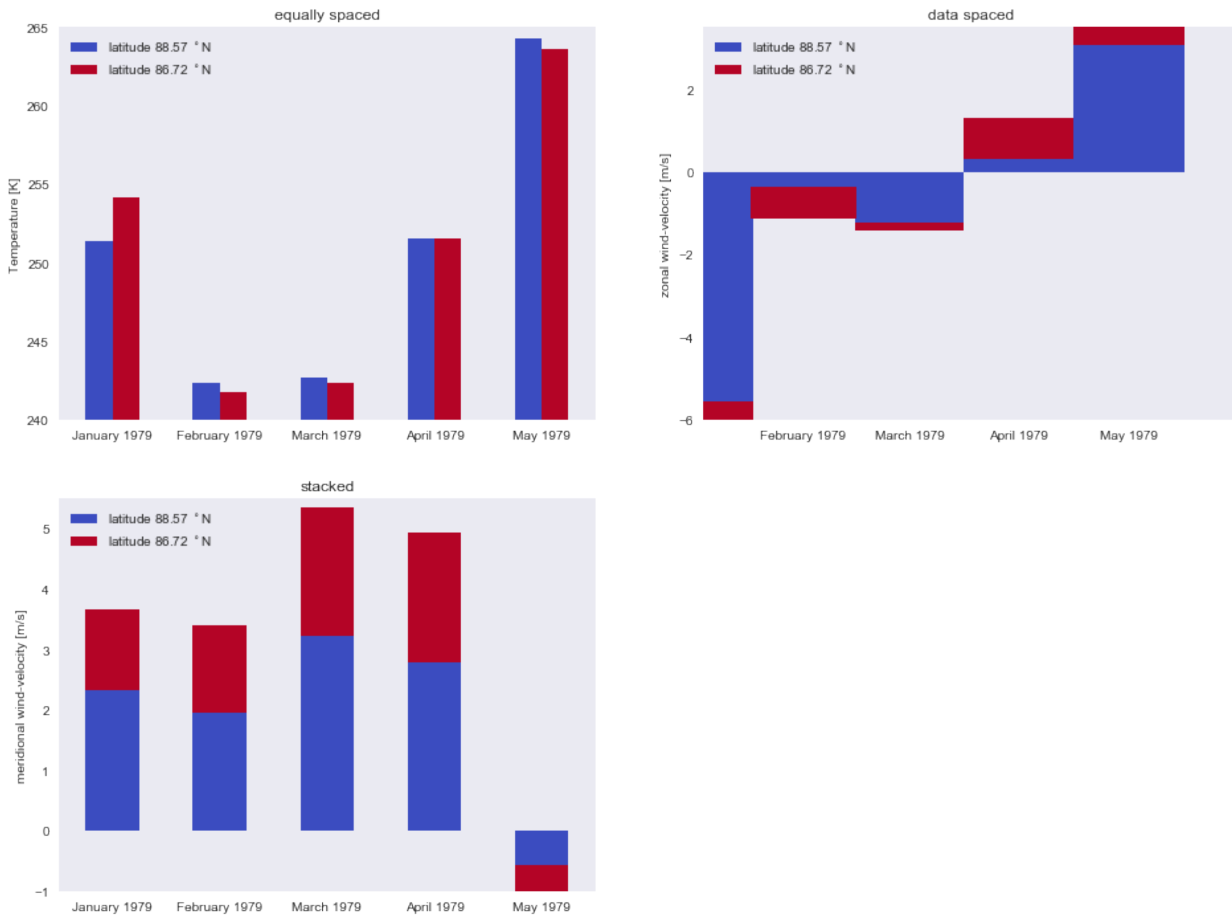
The default is that all bars have the same width. You can however change that by setting the `widths` keyword to `data`

```
bars(name='u').update(widths='data', xticks='month', title='data spaced')
bars.show()
```



Or you make a stacked plot

```
bars(name='v').update(plot='stacked', title='stacked')
bars.show()
```




```
psy.close('all')
```

1.3.3 2D plots

Demonstration of the 2D plot capabilities

The `plot2d` plot method make plots of 2-dimensional scalar data using matplotlibs `pcolormesh` or the `contourf` functions.

Note that this method is extended by the mapplot plot method of the psy-maps plugin for visualization on the projected globe.

```
import psyplot.project as psy
import xarray as xr
        import numpy as np
```

First we create some sample data in the form of a 2D parabola

```
x = np.linspace(-1, 1.)
y = np.linspace(-1, 1.)
x2d, y2d = np.meshgrid(x, y)
z = - x2d**2 - y2d**2
ds = xr.Dataset(
    {'z': xr.Variable(('x', 'y'), z)},
    {'x': xr.Variable(('x', ), x), 'y': xr.Variable(('y', ), y)})
```

For a simple 2D plot of a scalar field, we can use the plot2d plot method:

```
p = psy.plot.plot2d(ds, cmap='Reds', name='z')
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-4-20408b2849df> in <module>()
----> 1 p = psy.plot.plot2d(ds, cmap='Reds', name='z')


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in __call__(self, *args, **kwargs)
   1398         """
   1399         return self.project._add_data(
-> 1400             self.plotter_cls, *args, **dict(chain(
   1401                 [('prefer_list', self._prefer_list),
   1402                  ('default_slice', self._default_slice)],


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in plotter_cls(self)
   1345         if ret is None:
   1346             self.project.logger.debug('importing %s', self.module)
-> 1347             mod = import_module(self.module)
   1348             plotter = self.plotter_name
   1349             if plotter not in vars(mod):


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/__init__.py in import_module(name, package)
    124                 break
    125             level += 1
--> 126     return _bootstrap._gcd_import(name[level:], package, level)
    127
    128


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _gcd_import(name, package, level)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load(name, import_)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load_unlocked(name, import_)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _load_unlocked(spec)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap_external.py in exec_module(self, module)
```

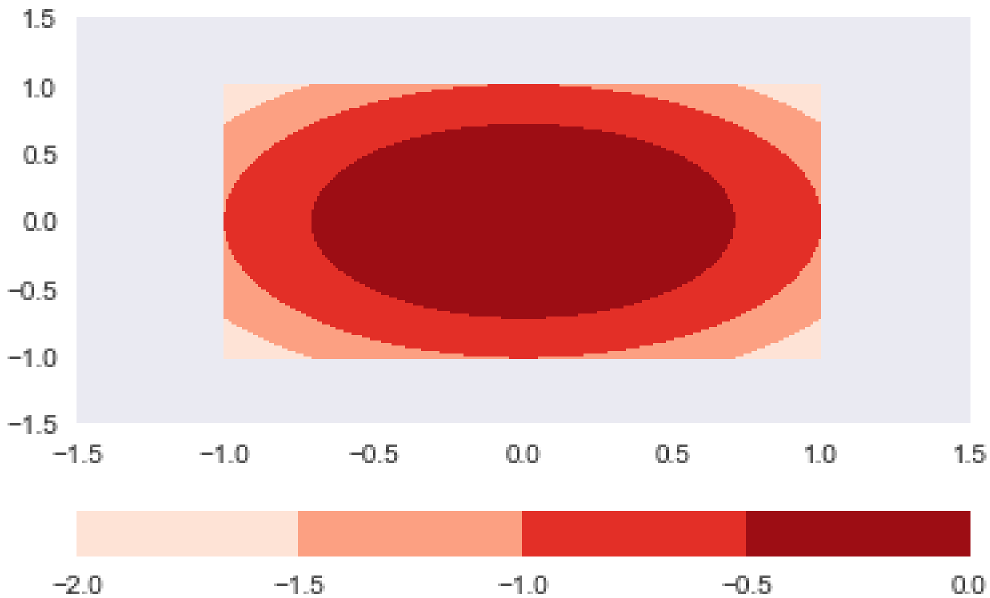
(continues on next page)

(continued from previous page)

```
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _call_with_frames_removed(f, *args, **kwds)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in <module>()
   5512
   5513
-> 5514 class FldmeanPlotter(LinePlotter):
   5515
   5516     _rcparams_string = ["plotter.fldmean."]


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in FldmeanPlotter()
   5523     # We reimplement the masking formatoption to make sure, that they are
   5524     # called after the mean calculation
-> 5525     maskgeq = MaskGeq('maskgeq', additional_children=['err_calc'])
   5526     maskleq = MaskLeq('maskleq', additional_children=['err_calc'])
   5527     maskgreater = MaskGreater('maskgreater', additional_children=['err_calc'])


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/plotter.py in __init__(self, key, plotter, index_in_list, **kwargs)
    429             raise TypeError(
    430                 '%s.__init__() got an unexpected keyword argument %r' % (
--> 431                     self.__class__.__name__, key))
    432         # set up child mapping
    433         self._child_mapping.update(kwargs)


TypeError: MaskGeq.__init__() got an unexpected keyword argument 'additional_children'
```

The `plot` formatoption controls, how the plot is made. The default is a pcolormesh plot, but we can also make a filled contour plot. The levels of the contour plot are determined through the `levels` formatoption.

```
p.update(plot='contourf', levels=5)
p.show()
```



The `plot2d` method has several formatoptions controlling the color coding of your plot:

```
p.keys('colors')
```

```
+-------------+-------------+-------------+-------------+
| levels      | miss_color  | cmap        | bounds      |
+-------------+-------------+-------------+-------------+
| extend      | cbar        | cbarspacing | cticksize   |
+-------------+-------------+-------------+-------------+
| ctickweight | ctickprops  |             |             |
+-------------+-------------+-------------+-------------+
```

The most important ones are

- `cbar`: To specify the location of the colorbar
- `bounds`: To specify the boundaries for the color coding, i.e. the categories which data range belongs to which color
- `cmap`: To specify the colormap

```
psy.close('all')
```

1.3.4 Violin plot demo

This example shows you how to make a violin plot using the `psyplot.project.ProjectPlotter.violinplot` method.

```
import psyplot.project as psy
```

```
axes = iter(psy.multiple_subplots(2, 2, n=3))
for var in ['t2m', 'u', 'v']:
    psy.plot.violinplot(
        'demo.nc',  # netCDF file storing the data
```

(continues on next page)

(continued from previous page)

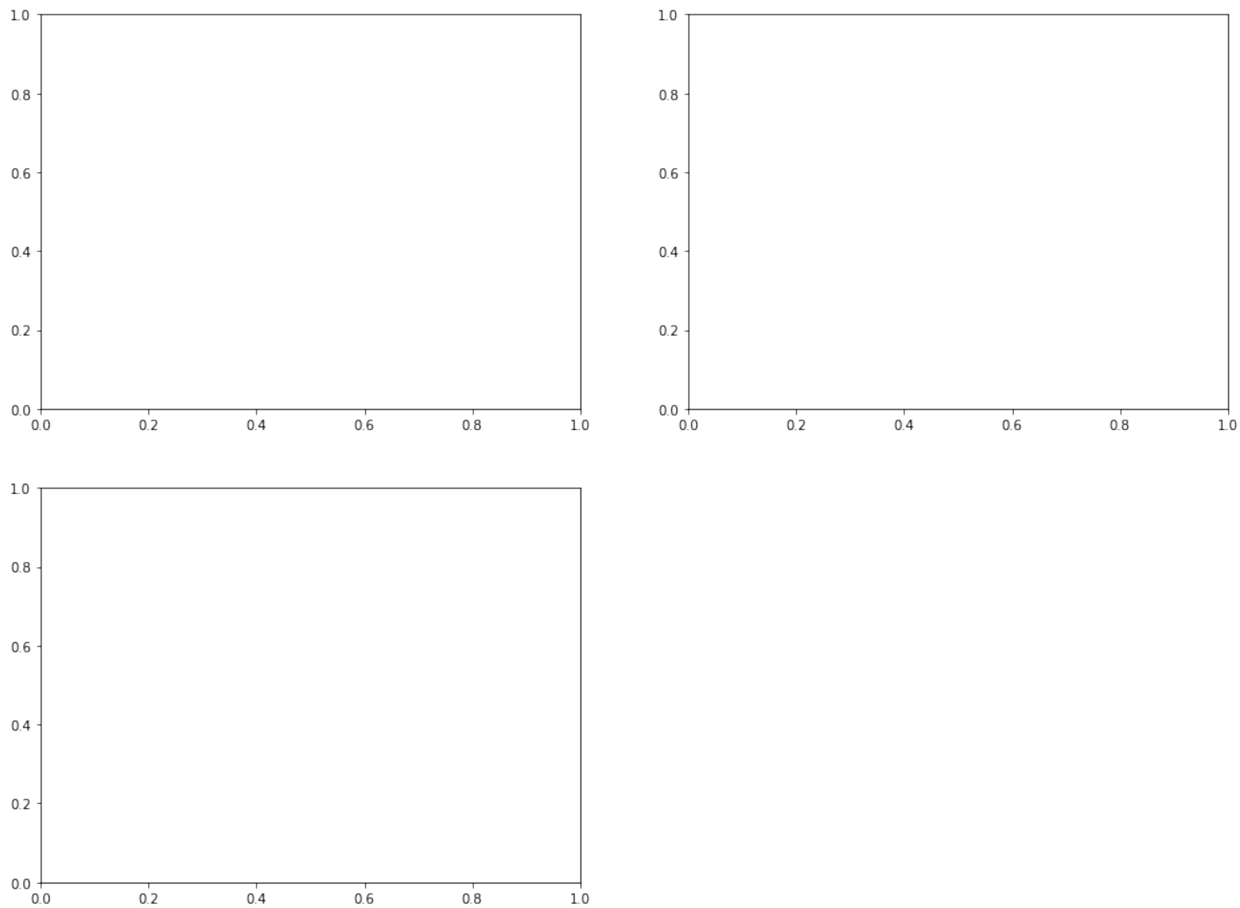
```
        name=var, # one plot for each variable
        t=range(5),  # one violin plot for each time step
        z=0, x=0,      # choose latitude and longitude as dimensions
        ylabel="{desc}",  # use the longname and units on the y-axis
        ax=next(axes),
        color='coolwarm', legend=False,
        xticklabels='%B %Y'  # choose xaxis labels to use month and year info,
    )
violins = psy.gcp(True)
violins.show()
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-3-166dc0c7a1f4> in <module>()
      9         ax=next(axes),
     10         color='coolwarm', legend=False,
---> 11         xticklabels='%B %Y'  # choose xaxis labels to use month and year info,
     12     )
     13 violins = psy.gcp(True)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in __call__(self, *args, **kwargs)
   1398         """
   1399         return self.project._add_data(
-> 1400             self.plotter_cls, *args, **dict(chain(
   1401                 [('prefer_list', self._prefer_list),
   1402                  ('default_slice', self._default_slice)],


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in plotter_cls(self)
   1345         if ret is None:
   1346             self.project.logger.debug('importing %s', self.module)
-> 1347             mod = import_module(self.module)
   1348             plotter = self.plotter_name
   1349             if plotter not in vars(mod):


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/__init__.py in import_module(name, package)
    124                 break
    125             level += 1
--> 126     return _bootstrap._gcd_import(name[level:], package, level)
    127
    128


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _gcd_import(name, package, level)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load(name, import_)
```

(continues on next page)

(continued from previous page)

```
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load_unlocked(name, import_)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _load_unlocked(spec)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap_external.py in exec_module(self, module)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _call_with_frames_removed(f, *args, **kwds)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in <module>()
   5512
   5513
-> 5514 class FldmeanPlotter(LinePlotter):
   5515
   5516     _rcparams_string = ["plotter.fldmean."]


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in FldmeanPlotter()
   5523     # We reimplement the masking formatoption to make sure, that they are
   5524     # called after the mean calculation
-> 5525     maskgeq = MaskGeq('maskgeq', additional_children=['err_calc'])
   5526     maskleq = MaskLeq('maskleq', additional_children=['err_calc'])
   5527     maskgreater = MaskGreater('maskgreater', additional_children=['err_calc'])


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/plotter.py in __init__(self, key, plotter, index_in_list, **kwargs)
    429             raise TypeError(
    430                 '%s.__init__() got an unexpected keyword argument %r' % (
--> 431                     self.__class__.__name__, key))
    432         # set up child mapping
    433         self._child_mapping.update(kwargs)


TypeError: MaskGeq.__init__() got an unexpected keyword argument 'additional_children'
```




```
psy.close('all')
```

1.3.5 Vector plot

Demonstration of the plotting of 2D vectors

The `vector` plotmethod uses matplotlibs `quiver` and `streamplot` functions to create the plot. This plot method requires two variables: `'u'` for the wind in `x`-direction, `'v'` for the wind in `y`-direction.

Note that this method is extended by the mapvector plot method of the psy-maps plugin for visualization on the projected globe.

```
import psyplot.project as psy
import xarray as xr
        import numpy as np
```

```
x2 = np.arange(0, 2 * np.pi, .2)
y2 = np.arange(0, 2 * np.pi, .2)
x22d, y22d = np.meshgrid(x2, y2)
ds2 = xr.Dataset(
    {'u': xr.Variable(('x', 'y'), np.cos(x22d)),
     'v': xr.Variable(('x', 'y'), np.sin(y22d))},
    {'x': xr.Variable(('x', ), x2),
     'y': xr.Variable(('y', ), y2)})
```

The default is a quiver plot

```
p = psy.plot.vector(ds2, name=[['u', 'v']], arrowsize=20.0)
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-4-bbde2fcb7496> in <module>()
----> 1 p = psy.plot.vector(ds2, name=[['u', 'v']], arrowsize=20.0)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in __call__(self, *args, **kwargs)
   1398         """
   1399         return self.project._add_data(
-> 1400             self.plotter_cls, *args, **dict(chain(
   1401                 [('prefer_list', self._prefer_list),
   1402                  ('default_slice', self._default_slice)],


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/project.py in plotter_cls(self)
   1345         if ret is None:
   1346             self.project.logger.debug('importing %s', self.module)
-> 1347             mod = import_module(self.module)
   1348             plotter = self.plotter_name
   1349             if plotter not in vars(mod):


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/__init__.py in import_module(name, package)
    124                 break
    125             level += 1
--> 126     return _bootstrap._gcd_import(name[level:], package, level)
    127
    128


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _gcd_import(name, package, level)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load(name, import_)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _find_and_load_unlocked(name, import_)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _load_unlocked(spec)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap_external.py in exec_module(self, module)
```

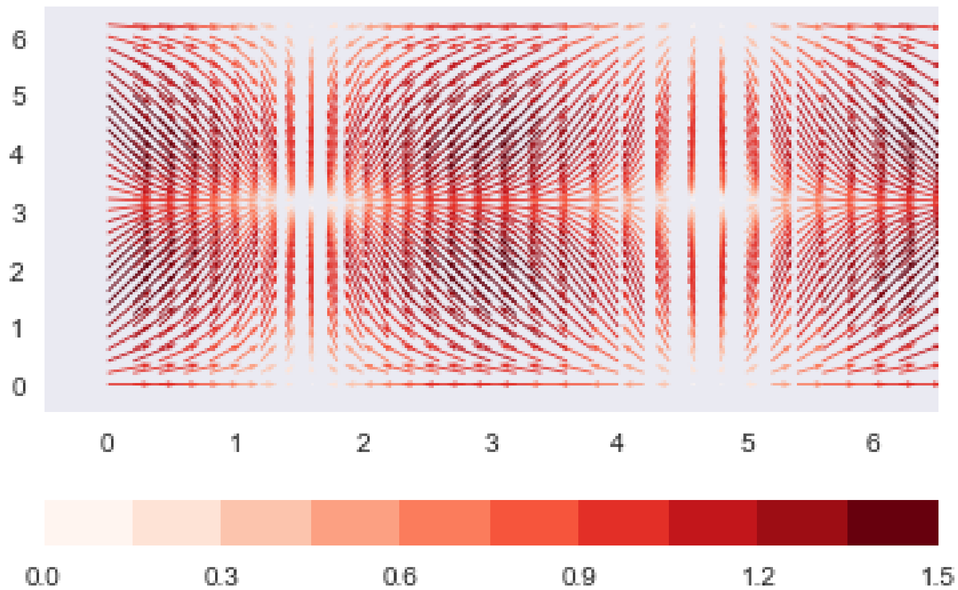
(continues on next page)

(continued from previous page)

```
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↪importlib/_bootstrap.py in _call_with_frames_removed(f, *args, **kwds)


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in <module>()
   5512
   5513
-> 5514 class FldmeanPlotter(LinePlotter):
   5515
   5516     _rcparams_string = ["plotter.fldmean."]


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in FldmeanPlotter()
   5523     # We reimplement the masking formatoption to make sure, that they are
   5524     # called after the mean calculation
-> 5525     maskgeq = MaskGeq('maskgeq', additional_children=['err_calc'])
   5526     maskleq = MaskLeq('maskleq', additional_children=['err_calc'])
   5527     maskgreater = MaskGreater('maskgreater', additional_children=['err_calc'])


~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↪packages/psyplot/plotter.py in __init__(self, key, plotter, index_in_list, **kwargs)
    429             raise TypeError(
    430                 '%s.__init__() got an unexpected keyword argument %r' % (
--> 431                     self.__class__.__name__, key))
    432         # set up child mapping
    433         self._child_mapping.update(kwargs)


TypeError: MaskGeq.__init__() got an unexpected keyword argument 'additional_children'
```

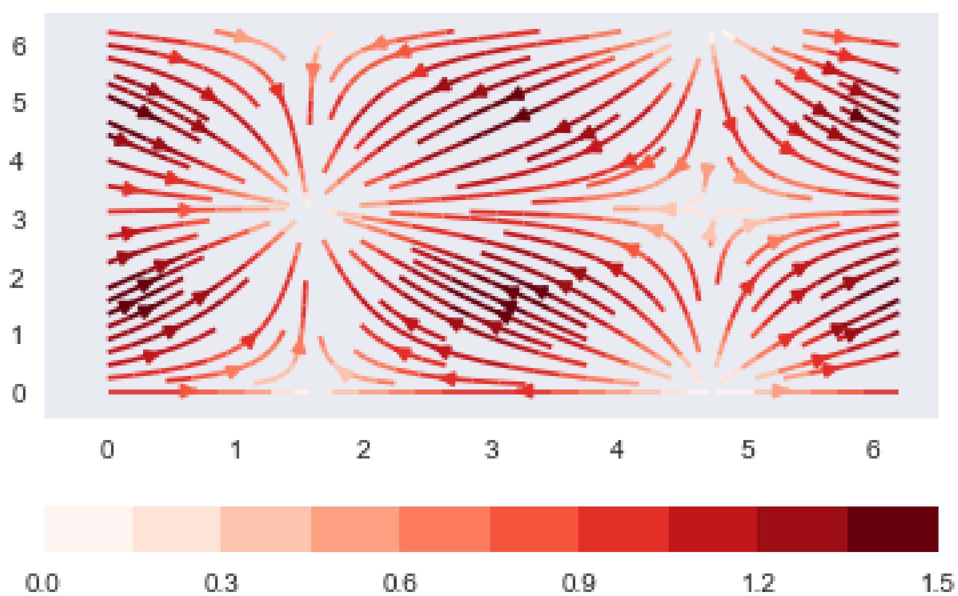
You can also apply a colormap to the vectors via the `colors` formatoption. This can be one on `'u'` (to use the x-direction), `'v'` (to use the `'y'`-direction) or `'absolute'` to use the absolute length of the arrows.

```
p.update(cmap='Reds', color='absolute')
p.show()
```



The `vector` plot method also supports stream plots through the `plot` formatoption.

```
p.update(plot='stream', arrowsize=1.0)
p.show()
```



The main formatoptions for the vector plots are in the `vector` group

```
p.summaries('vector')
```

```
arrowsize
    Change the size of the arrows
arrowstyle
    Change the style of the arrows
density
```

(continues on next page)

(continued from previous page)

```
    Change the density of the arrows
```

```
psy.close('all')
```

1.4 API Reference

psy-simple: The psyplot plugin for simple visualizations

This package contains the basic plotters for simple interactive visualization tasks with the psyplot visualization framework.

1.4.1 Subpackages

psy_simple.widgets package

Submodules

psy_simple.widgets.colors module

psy_simple.widgets.texts module

1.4.2 Submodules

psy_simple.base module

Plotter classes

BasePlotter([data, ax, auto_update, ...])	Base class with formatoptions for plotting on an matplotlib axes
TitlesPlotter([data, ax, auto_update, ...])	Plotter class for labels

Formatoption classes

Figtitle(key[, plotter, index_in_list, ...])	Plot a figure title
MaskBetween(key[, plotter, index_in_list, ...])	Mask data points between two numbers
MaskGeq(key[, plotter, index_in_list, ...])	Mask data points greater than or equal to a number
MaskGreater(key[, plotter, index_in_list, ...])	Mask data points greater than a number
MaskLeq(key[, plotter, index_in_list, ...])	Mask data points smaller than or equal to a number
MaskLess(key[, plotter, index_in_list, ...])	Mask data points smaller than a number
Text(*args, **kwargs)	Add text anywhere on the plot
Tight(key[, plotter, index_in_list, ...])	Automatically adjust the plots.
Title(key[, plotter, index_in_list, ...])	Show the title
ValueMaskBase(key[, plotter, index_in_list, ...])	Base class for masking formatoptions

Classes

`TextBase`	Abstract base class for formatoptions that provides a replace method

Functions

`label_props`(base[, label_name, children, . . .])	Function that returns a Formatoption class for modifying the fontsite
`label_size`(base[, label_name, children, . . .])	Function that returns a Formatoption class for modifying the fontsite
`label_weight`(base[, label_name, children, . . .])	Function that returns a Formatoption class for modifying the fontweight

class psy_simple.base.**BasePlotter**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: `psy_simple.base.TitlesPlotter`

Base class with formatoptions for plotting on an matplotlib axes

Parameters

- **data** (`InteractiveArray or ArrayList, optional`) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (`matplotlib.axes.Axes`) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (`bool`) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (`bool or None`) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (`bool`) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (`bool`) – If True, the axes is cleared first
- **enable_post** (`bool`) – If True, the `post` formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Masking formatoptions

`maskbetween`	Mask data points between two numbers
`maskgeq`	Mask data points greater than or equal to a number
`maskgreater`	Mask data points greater than a number
`maskleq`	Mask data points smaller than or equal to a number
`maskless`	Mask data points smaller than a number

Axes formatoptions

tight	Automatically adjust the plots.

Label formatoptions

figtitle	Plot a figure title
figtitleprops	Properties of the figure title
figtitlesize	Set the size of the figure title
figtitleweight	Set the fontweight of the figure title
text	Add text anywhere on the plot
title	Show the title
titleprops	Properties of the title
titlesize	Set the size of the title
titleweight	Set the fontweight of the title

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

maskbetween
Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskgeq*

maskgeq
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgeq*, *maskbetween*

maskleq
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

tight
Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

Warning: There is no update method to undo what happend after this formatoption is set to True!

figtitle
Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`

- tinfo: `%H:%M`
- sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this *`Figtitle`* instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *`title`* formatoption.

See also:

`title`, *`figtitlesize`*, *`figtitleweight`*, *`figtitleprops`*

`figtitleprops`
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`figtitle`, *`figtitlesize`*, *`figtitleweight`*

`figtitlesize`
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`figtitle`, *`figtitleweight`*, *`figtitleprops`*

`figtitleweight`
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

figtitle, *figtitlesize*, *figtitleprops*

text
Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

title, *figtitle*

title
Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the *`figtitle`* formatoption.

See also:

`figtitle`, *`titlesize`*, *`titleweight`*, *`titleprops`*

titleprops
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`title`, *`titlesize`*, *`titleweight`*

titlesize
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

title, *titleweight*, *titleprops*

titleweight

Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

title, *titlesize*, *titleprops*

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

class psy_simple.base.**Figtitle**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.base.TextBase*, psyplot.plotter.Formatoption

Plot a figure title

Set the title of the figure. You can insert any meta key from the xarray.DataArray.attrs via a string like '%(key)s'. Furthermore there are some special cases:

- Strings like '%Y', '%b', etc. will be replaced using the datetime.datetime.strftime() method as long as the data has a time coordinate and this can be converted to a datetime object.
- '%(x)s', '%(y)s', '%(z)s', '%(t)s' will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via axis + key (e.g. the name of the x-coordinate can be inserted via '%(xname)s').
- Labels defined in the psyplot.rcParams 'texts.labels' key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: %B %d, %Y. %H:%M
  - desc: %(long_name)s [%(units)s]
  - dinfo: %B %d, %Y
  - tinfo: %H:%M
  - sdesc: %(name)s [%(units)s]

Methods

clear_other_texts([remove])	Make sure that no other text is a the same position as this one
initialize_plot(s)	Method that is called when the plot is made the first time
update(s)	Method that is call to update the formatoption on the axes

Attributes

enhanced_attrs	The enhanced attributes of the array
name	str(object='') -> str

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this *Figtitle* instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the `title` formatoption.

See also:

`title`, `figtitlesize`, `figtitleweight`, `figtitleprops`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

clear_other_texts(*remove=False*)

Make sure that no other text is a the same position as this one

This method clears all text instances in the figure that are at the same position as the `_text` attribute

Parameters **remove** (*bool*) – If True, the Text instances are permanently deleted from the figure, otherwise there text is simply set to ‘’

enhanced_attrs
The enhanced attributes of the array

initialize_plot(*s*)
Method that is called when the plot is made the first time

Parameters **value** – The value to use for the initialization

name = 'Figure title'

update(*s*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.base.**MaskBetween**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: *psy_simple.base.ValueMaskBase*

Mask data points between two numbers

Possible types

Methods

mask_func(data, value)	The masking function that is called

Attributes

name	str(object=’‘) -> str

float – The floating number to mask above

See also:

maskless, maskleq, maskgreater, maskgeq

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the dependencies attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords

may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

mask_func (*data*, *value*)
The masking function that is called

name = 'Mask between two values'

class psy_simple.base.**MaskGeq** (*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psy_simple.base.ValueMaskBase`

Mask data points greater than or equal to a number

Possible types

Methods

`mask_func`(data, value)	The masking function that is called

Attributes

`name`	str(object='') -> str

float – The floating number to mask above

See also:

`maskless`, `maskleq`, `maskgreater`, `maskbetween`

Parameters

- **key** (`str`) – formatoption key in the *plotter*
- **plotter** (`psyplot.plotter.Plotter`) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (`int or None`) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (`list or str`) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (`list or str`) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

mask_func (*data*, *value*)
The masking function that is called

name = 'Mask greater than or equal'

class psy_simple.base.**MaskGreater**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.base.ValueMaskBase*

Mask data points greater than a number

Possible types

Methods

mask_func(data, value)	The masking function that is called

Attributes

name	str(object='') -> str

float – The floating number to mask above

See also:

maskless, maskleq, maskgeq, maskbetween

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the dependencies attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the children, dependencies and connections attributes, with values being the name of the new formatoption in this plotter.

mask_func(*data*, *value*)

The masking function that is called

name = 'Mask greater'

class psy_simple.base.**MaskLeq**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.base.ValueMaskBase*

Mask data points smaller than or equal to a number

Possible types

Methods

mask_func(data, value)	The masking function that is called

Attributes

name	str(object=’‘) -> str

float – The floating number to mask below

See also:

`maskless`, `maskgreater`, `maskgeq`, `maskbetween`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

mask_func(*data*, *value*)
The masking function that is called

name = 'Mask lesser than or equal'

class psy_simple.base.**MaskLess**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: *psy_simple.base.ValueMaskBase*

Mask data points smaller than a number

Possible types

Methods

mask_func(data, value)	The masking function that is called

Attributes

name	str(object='') -> str

float – The floating number to mask below

See also:

maskleq, maskgreater, maskgeq, maskbetween

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the dependencies attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the children, dependencies and connections attributes, with values being the name of the new formatoption in this plotter.

mask_func(*data*, *value*)
The masking function that is called

name = 'Mask less'

class psy_simple.base.**Text**(**args*, ***kwargs*)
Bases: *psy_simple.base.TextBase*, psyplot.plotter.Formatoption

Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the xarray.DataArray.attrs via a string like '%(key)s'. Furthermore there are some special cases:

- Strings like '%Y', '%b', etc. will be replaced using the datetime.datetime.strftime() method as long as the data has a time coordinate and this can be converted to a datetime object.
- '%(x)s', '%(y)s', '%(z)s', '%(t)s' will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via axis + key (e.g. the name of the x-coordinate can be inserted via '%(xname)s').
- Labels defined in the psyplot.rcParams 'texts.labels' key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: %B %d, %Y. %H:%M
  - desc: %(long_name)s [%(units)s]
  - dinfo: %B %d, %Y
  - tinfo: %H:%M
  - sdesc: %(name)s [%(units)s]

Methods

diff(value)	Checks whether the given value differs from what is currently set
finish_update()	Clears the `_texts_to_remove` set
remove()	Method to remove the effects of this formatoption
set_value(value[, validate, todefault])	Set (and validate) the value in the plotter.
share(fmto, **kwargs)	Share the settings of this formatoption with other data objects
update(value[, texts_to_remove])	Method that is call to update the formatoption on the axes

Attributes

name	str(object='') -> str
transform	Dictionary containing the relevant transformations

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

`title`, `figtitle`

diff(*value*)

Checks whether the given value differs from what is currently set

Parameters value – A possible value to set (make sure that it has been validate via the `validate` attribute before)

Returns True if the value differs from what is currently set

Return type bool

finish_update()

Clears the `_texts_to_remove` set

name = 'Arbitrary text on the plot'

remove()

Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

set_value (*value*, *validate=True*, *todefault=False*)
Set (and validate) the value in the plotter. This method is called by the plotter when it attempts to change the value of the formatoption.

Parameters

- **value** – Value to set
- **validate** (*bool*) – if True, validate the *value* before it is set
- **todefault** (*bool*) – True if the value is updated to the default value

share (*fmto*, ***kwargs*)
Share the settings of this formatoption with other data objects

Parameters

- **fmto** (*Formatoption*) – The `Formatoption` instance to share the attributes with
- ****kwargs** – Any other keyword argument that shall be passed to the update method of *fmto*

Notes

The Text formatoption sets the 'texts_to_remove' keyword to the `_texts_to_remove` attribute of this instance (if not already specified in `**kwargs`

transform
Dictionary containing the relevant transformations

update (*value*, *texts_to_remove=None*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.base.**TextBase**
Bases: `object`

Abstract base class for formatoptions that provides a replace method **Attributes**

data_dependent	bool(x) -> bool
enhanced_attrs	The enhanced attributes of the array
group	str(object='') -> str
rc	`SubDict` of rcParams 'texts' key

Methods

get_enhanced_attrs(*args, **kwargs)	
get_fig_data_attrs([delimiter])	Join the data attributes with other plotters in the project
get_fmt_widget(parent, project)	Create a combobox with the attributes
replace(s, data[, attrs])	Replace the attributes of the plotter data in a string

data_dependent = True

delimiter = None

enhanced_attrs
The enhanced attributes of the array

get_enhanced_attrs (**args*, ***kwargs*)

get_fig_data_attrs (*delimiter=None*)
Join the data attributes with other plotters in the project

This method joins the attributes of the `InteractiveBase` instances in the project that draw on the same figure as this instance does.

Parameters **delimiter** (*str*) – Specifies the delimiter with what the attributes are joined. If None, the *delimiter* attribute of this instance or (if the latter is also None), the rcParams['texts.delimiter'] item is used.

Returns A dictionary with all the meta attributes joined by the specified *delimiter*

Return type dict

get_fmt_widget (*parent, project*)
Create a combobox with the attributes

group = 'labels'

rc
`SubDict` of rcParams 'texts' key

replace (*s, data, attrs=None*)
Replace the attributes of the plotter data in a string

You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Parameters

- **s** (*str*) – String where the replacements shall be made
- **data** (*InteractiveBase*) – Data object from which to use the coordinates and insert the coordinate and attribute informations
- **attrs** (*dict*) – Meta attributes that shall be used for replacements. If None, it will be gained from *data.attrs*

Returns *s* with inserted informations

Return type str

class psy_simple.base.**Tight**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

Attributes

group	str(object='') -> str
name	str(object='') -> str

Methods

update(value)	Method that is call to update the formatoption on the axes

bool – True for automatic adjustment

Warning: There is no update method to undo what happend after this formatoption is set to True!

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

group = 'axes'

name = 'Tight layout'

update(*value*)

Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.base.**Title**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.base.TextBase*, psyplot.plotter.Formatoption

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Methods

initialize_plot(value)	Method that is called when the plot is made the first time
update(value)	Method that is call to update the formatoption on the axes

Attributes

name	str(object='') -> str

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the `figtitle` formatoption.

See also:

`figtitle`, `titlesize`, `titleweight`, `titleprops`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.

- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

initialize_plot(*value*)
Method that is called when the plot is made the first time

Parameters value – The value to use for the initialization

name = 'Axes title'

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.base.**TitlesPlotter**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: `psyplot.plotter.Plotter`

Plotter class for labels

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the `post` formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Label formatoptions

figtitle	Plot a figure title
figtitleprops	Properties of the figure title
figtitlesize	Set the size of the figure title
figtitleweight	Set the fontweight of the figure title
text	Add text anywhere on the plot
title	Show the title
titleprops	Properties of the title
titlesize	Set the size of the title
titleweight	Set the fontweight of the title

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

figtitle

Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this *Figtitle* instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *title* formatoption.

See also:

title, *figtitlesize*, *figtitleweight*, *figtitleprops*

figtitleprops
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

figtitle, *figtitlesize*, *figtitleweight*

figtitlesize
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

figtitle, *figtitleweight*, *figtitleprops*

figtitleweight
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

figtitle, *figtitlesize*, *figtitleprops*

text
Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).

- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

`title`, *`figtitle`*

`title`

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the `figtitle` formatoption.

See also:

`figtitle`, `titlesize`, `titleweight`, `titleprops`

titleprops
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`title`, `titlesize`, `titleweight`

titlesize
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be ‘xx-small’, ‘x-small’, ‘small’, ‘medium’, ‘large’, ‘x-large’, ‘xx-large’.

See also:

`title`, `titleweight`, `titleprops`

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of ‘ultralight’, ‘light’, ‘normal’, ‘regular’, ‘book’, ‘medium’, ‘roman’, ‘semibold’, ‘demibold’, ‘demi’, ‘bold’, ‘heavy’, ‘extra bold’, ‘black’.

See also:

`title`, `titlesize`, `titleprops`

post
Apply your own postprocessing script

This formatoption let’s you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

class psy_simple.base.**ValueMaskBase**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: psyplot.plotter.Formatoption

Base class for masking formatoptions

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

Attributes

data_dependent	bool(x) -> bool
group	str(object='') -> str
priority	int(x=0) -> integer

Methods

mask_func()	The masking function that is called
update(value)	Method that is call to update the formatoption on the axes

data_dependent = True

group = 'masking'

mask_func()

The masking function that is called

priority = 30

update(*value*)

Method that is call to update the formatoption on the axes

Parameters value – Value to update

psy_simple.base.**label_props**(*base*, *label_name=None*, *children=[]*, *parents=[]*, *dependencies=[]*)

Function that returns a Formatoption class for modifying the fontsite

This function returns a Formatoption instance that modifies the size of the given *base* formatoption

Parameters

- **base** (*Formatoption*) – The base formatoption instance that is used in the `psyplot.Plotter` subclass to create the label. The instance must have a `texts` attribute which stores all the `matplotlib.text.Text` instances.
- **label_name** (*str*) – The name of the label to use in the documentation. If None, it will be `key`, where `key` is the `psyplot.plotter.Formatoption.key`` attribute of *base*
- **children** (*list of str*) – The childrens of the resulting formatoption class (besides the *base* formatoption which is included anyway)
- **parents** (*list of str*) – The parents of the resulting formatoption class (besides the *base* the properties formatoption from *base* (see `label_props()`))
- **dependencies** (*list of str*) – The dependencies of the formatoption
- **children** – The childrens of the resulting formatoption class (besides the *base* formatoption, the `base.key + 'size'` and `base.key + 'weight'` keys, which are included anyway (see `label_size()`, `label_weight()`))
- **parents** – The parents of the resulting formatoption class

Returns The formatoption instance that modifies the fontsize of *base*

Return type Formatoption

See also:

`label_weight()`, `label_props()`, `Figtitle()`, `Title()`

`psy_simple.base.`**`label_size`**(*base*, *label_name=None*, *children=[]*, *parents=[]*, *dependencies=[]*)

Function that returns a Formatoption class for modifying the fontsite

This function returns a `Formatoption` instance that modifies the size of the given *base* formatoption

Parameters

- **base** (*Formatoption*) – The base formatoption instance that is used in the `psyplot.Plotter` subclass to create the label. The instance must have a `texts` attribute which stores all the `matplotlib.text.Text` instances.
- **label_name** (*str*) – The name of the label to use in the documentation. If None, it will be `key`, where `key` is the `psyplot.plotter.Formatoption.key`` attribute of *base*
- **children** (*list of str*) – The childrens of the resulting formatoption class (besides the *base* formatoption which is included anyway)
- **parents** (*list of str*) – The parents of the resulting formatoption class (besides the *base* the properties formatoption from *base* (see `label_props()`))
- **dependencies** (*list of str*) – The dependencies of the formatoption

Returns The formatoption instance that modifies the fontsize of *base*

Return type Formatoption

See also:

`label_weight()`, `label_props()`, `Figtitle()`, `Title()`

`psy_simple.base.`**`label_weight`**(*base*, *label_name=None*, *children=[]*, *parents=[]*, *dependencies=[]*)

Function that returns a Formatoption class for modifying the fontweight

This function returns a `Formatoption` instance that modifies the weight of the given *base* formatoption

Parameters

- **base** (*Formatoption*) – The base formatoption instance that is used in the `psyplot.Plotter` subclass to create the label. The instance must have a `texts` attribute which stores all the `matplotlib.text.Text` instances.
- **label_name** (*str*) – The name of the label to use in the documentation. If None, it will be `key`, where `key` is the `psyplot.plotter.Formatoption.key`` attribute of *base*
- **children** (*list of str*) – The childrens of the resulting formatoption class (besides the *base* formatoption which is included anyway)
- **parents** (*list of str*) – The parents of the resulting formatoption class (besides the *base* the properties formatoption from *base* (see `label_props()`))
- **dependencies** (*list of str*) – The dependencies of the formatoption

Returns The formatoption instance that modifies the fontweight of *base*

Return type Formatoption

See also:

`label_size()`, `label_props()`, `Figtitle()`, `Title()`

psy_simple.colors module

colors module of the psyplot package.

This module contains some additional color maps and the show_colormaps function to visualize available colormaps.

Classes

`FixedBoundaryNorm`(boundaries, ncolors[, clip])	Bug fixing Norm with same functionality as matplotlibs BoundaryNorm
`FixedColorMap`(name, segmentdata[, N, gamma])	Bug fixing colormap with same functionality as matplotlibs colormap

Functions

`get_cmap`(name[, lut])	Returns the specified colormap.
`show_colormaps`([names, N, show, use_qt])	Function to show standard colormaps from pyplot

class psy_simple.colors.**FixedBoundaryNorm**(*boundaries*, *ncolors*, *clip=False*)

Bases: `matplotlib.colors.BoundaryNorm`

Bug fixing Norm with same functionality as matplotlibs BoundaryNorm

This class fixes a bug in the `cartopy.mpl.geoaxes.GeoAxes.streamplot()` for matplotlib version 1.5

Notes

To reproduce the error type:

```
>>> import cartopy.crs as ccrs
>>> import matplotlib.pyplot as plt
>>> import psyplot.project as psy
>>> import matplotlib.colors as mcol
>>> maps = psy.plot.mapvector(
...     'test-t2m-u-v.nc', name=[['u', 'v']], plot='stream',
...     lonlatbox='Europe', color='absolute')
>>> plotter = maps[0].plotter
>>> x, y, u, v = plotter.plot._get_data()
>>> maps.close(True, True)
>>> ax = plt.axes(projection=ccrs.PlateCarree())
>>> ax.set_extent(plotter.lonlatbox.lonlatbox, crs=ccrs.PlateCarree())
>>> m = ax.streamplot(
...     x, y, u, v, color=plotter.plot._kwargs['color'],
...     norm=mcol.BoundaryNorm(plotter.bounds.norm.boundaries,
...                            plotter.bounds.norm.Ncmap,
...                            plotter.bounds.norm.clip),
...                            density=[1.0, 1.0])
```

This raises in matplotlib.colors, line 1316, in `matplotlib.colors.BoundaryNorm.__call__()`:

```
``ret = int(ret[0])  # assume python scalar``
MaskError: Cannot convert masked element to a Python int.
```

Parameters

- **boundaries** (*array-like*) – Monotonically increasing sequence of boundaries
- **ncolors** (*int*) – Number of colors in the colormap to be used
- **clip** (*bool, optional*) – If clip is `True`, out of range values are mapped to 0 if they are below `boundaries[0]` or mapped to ncolors - 1 if they are above `boundaries[-1]`.

  If clip is `False`, out of range values are mapped to -1 if they are below `boundaries[0]` or mapped to ncolors if they are above `boundaries[-1]`. These are then converted to valid indices by `Colormap.__call__()`.

Notes

boundaries defines the edges of bins, and data falling within a bin is mapped to the color with the same index.

If the number of bins doesn't equal *ncolors*, the color is chosen by linear interpolation of the bin number onto color numbers.

class `psy_simple.colors.`**`FixedColorMap`**(*name, segmentdata, N=256, gamma=1.0*)

Bases: `matplotlib.colors.LinearSegmentedColormap`

Bug fixing colormap with same functionality as matplotlibs colormap

This class fixes a bug in the `cartopy.mpl.geoaxes.GeoAxes.streamplot()` method in python 3.4

Notes

Methods

from_list(**kwargs)	Make a linear segmented colormap with *name* from a sequence of *colors* which evenly transitions from colors[0] at val=0 to colors[-1] at val=1.

To reproduce the error type in python 3.4:

```
>>> import cartopy.crs as ccrs
>>> import matplotlib.pyplot as plt
>>> import psyplot.project as psy
>>> maps = psy.plot.mapvector(
...     'test-t2m-u-v.nc', name=[['u', 'v']], plot='stream',
...     lonlatbox='Europe', color='absolute')
>>> plotter = maps[0].plotter
>>> x, y, u, v = plotter.plot._get_data()
>>> maps.close(True, True)
>>> ax = plt.axes(projection=ccrs.PlateCarree())
>>> ax.set_extent(plotter.lonlatbox.lonlatbox, crs=ccrs.PlateCarree())
>>> m = ax.streamplot(x, y, u, v, density=[1.0, 1.0],
...                   color=plotter.plot._kwargs['color'],
...                   norm=plotter.plot._kwargs['norm'])
```

This raises in matplotlib.colors, line 557, in `matplotlib.colors.Colormap.__call__()`:

```
``xa = np.array([X])``
ValueError: setting an array element with a sequence.
```

Create color map from linear mapping segments

segmentdata argument is a dictionary with a red, green and blue entries. Each entry should be a list of *x*, *y0*, *y1* tuples, forming rows in a table. Entries for alpha are optional.

Example: suppose you want red to increase from 0 to 1 over the bottom half, green to do the same over the middle half, and blue over the top half. Then you would use:

```
cdict = {'red':   [(0.0,  0.0, 0.0),
                   (0.5,  1.0, 1.0),
                   (1.0,  1.0, 1.0)],

         'green': [(0.0,  0.0, 0.0),
                   (0.25, 0.0, 0.0),
                   (0.75, 1.0, 1.0),
                   (1.0,  1.0, 1.0)],

         'blue':  [(0.0,  0.0, 0.0),
                   (0.5,  0.0, 0.0),
                   (1.0,  1.0, 1.0)]}
```

Each row in the table for a given color is a sequence of *x*, *y0*, *y1* tuples. In each sequence, *x* must increase monotonically from 0 to 1. For any input value *z* falling between *x[i]* and *x[i+1]*, the output value of a given color will be linearly interpolated between *y1[i]* and *y0[i+1]*:

```
row i:   x  y0  y1
               /
              /
row i+1: x  y0  y1
```

Hence y0 in the first row and y1 in the last row are never used.

See also:

`LinearSegmentedColormap.from_list()` Static method; factory function for generating a smoothly-varying LinearSegmentedColormap.

`makeMappingArray()` For information about making a mapping array.

static from_list(***kwargs*)
Make a linear segmented colormap with *name* from a sequence of *colors* which evenly transitions from colors[0] at val=0 to colors[-1] at val=1. *N* is the number of rgb quantization levels. Alternatively, a list of (value, color) tuples can be given to divide the range unevenly.

psy_simple.colors.**get_cmap**(*name*, *lut=None*)
Returns the specified colormap.

Parameters

- **name** (str or `matplotlib.colors.Colormap`) – If a colormap, it returned unchanged.

  Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).

- **lut** (*int*) – An integer giving the number of entries desired in the lookup table

Returns The colormap specified by *name*

Return type matplotlib.colors.Colormap

See also:

show_colormaps() A function to display all available colormaps

Notes

Different from the :func::*matpltolib.pyplot.get_cmap* function, this function changes the number of colors if *name* is a `matplotlib.colors.Colormap` instance to match the given *lut*.

psy_simple.colors.**show_colormaps**(*names=[]*, *N=10*, *show=True*, *use_qt=None*)
Function to show standard colormaps from pyplot

Parameters

- ***args** (str or `matplotlib.colors.Colormap`) – If a colormap, it returned unchanged.

  Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).

- **N** (*int, optional*) – Default: 11. The number of increments in the colormap.
- **show** (*bool, optional*) – Default: True. If True, show the created figure at the end with pyplot.show(block=False)
- **use_qt** (*bool*) – If True, use the `psy_simple.widgets.color.ColormapDialog.show_colormaps`, if False use a matplotlib implementation based on[1]. If None, use the Qt implementation if it is running in the psyplot GUI.

[1] http://matplotlib.org/1.2.1/examples/pylab_examples/show_colormaps.html

Returns Depending on *use_qt*, either an instance of the `psy_simple.widgets.color.ColormapDialog` or the `matplotlib.figure.Figure`

Return type psy_simple.widgets.color.ColormapDialog or matplitlib.figure.Figure

References

psy_simple.plotters module

Formatoption classes

`AlternativeXCoord`(key[, plotter, ...])	Use an alternative variable as x-coordinate
`AlternativeXCoordPost`(key[, plotter, ...])	Use an alternative variable as x-coordinate
`ArrowSize`(key[, plotter, index_in_list, ...])	Change the size of the arrows
`ArrowStyle`(key[, plotter, index_in_list, ...])	Change the style of the arrows
`AxisColor`(key[, plotter, index_in_list, ...])	Color the x- and y-axes
`BarAlpha`(key[, plotter, index_in_list, ...])	Specify the transparency (alpha)
`BarPlot`(*args, **kwargs)	Choose how to make the bar plot
`BarWidths`(key[, plotter, index_in_list, ...])	Specify the widths of the bars
`BarXTickLabels`(*args, **kwargs)	Modify the x-axis ticklabels
`BarXlabel`(key[, plotter, index_in_list, ...])	Set the x-axis label
`BarXlim`(*args, **kwargs)	Set the x-axis limits
`BarYTickLabels`(*args, **kwargs)	Modify the y-axis ticklabels
`BarYlabel`(key[, plotter, index_in_list, ...])	Set the y-axis label
`BarYlim`(*args, **kwargs)	Set the y-axis limits
`Bounds`(*args, **kwargs)	Specify the boundaries of the colorbar
`CLabel`(key[, plotter, index_in_list, ...])	Show the colorbar label
`CMap`(key[, plotter, index_in_list, ...])	Specify the color map
`CTickLabels`(*args, **kwargs)	Specify the colorbar ticklabels
`CTickProps`(key[, plotter, index_in_list, ...])	Specify the font properties of the colorbar ticklabels
`CTickSize`(key[, plotter, index_in_list, ...])	Specify the font size of the colorbar ticklabels
`CTickWeight`(key[, plotter, index_in_list, ...])	Specify the fontweight of the colorbar ticklabels
`CTicks`(*args, **kwargs)	Specify the tick locations of the colorbar
`Cbar`(*args, **kwargs)	Specify the position of the colorbars
`CbarOptions`(key[, plotter, index_in_list, ...])	Base class for colorbar formatoptions
`CbarSpacing`(key[, plotter, index_in_list, ...])	Specify the spacing of the bounds in the colorbar
`CombinedVectorPlot`(*args, **kwargs)	Choose the vector plot type
`ContourLevels`(*args, **kwargs)	The levels for the contour plot
`DataGrid`(key[, plotter, index_in_list, ...])	Show the grid of the data
`DataPrecision`(key[, plotter, index_in_list, ...])	Set the precision of the data
`DataTicksCalculator`(*args, **kwargs)	Abstract base formatoption to calculate ticks and bounds from the data
`Density`(*args, **kwargs)	Change the density of the arrows
`DtTicksBase`(*args, **kwargs)	Abstract base class for x- and y-tick formatoptions
`ErrorAlpha`(key[, plotter, index_in_list, ...])	Set the alpha value for the error range
`ErrorCalculator`(key[, plotter, ...])	Calculation of the error
`ErrorPlot`(*args, **kwargs)	Visualize the error range
`Extend`(key[, plotter, index_in_list, ...])	Draw arrows at the side of the colorbar
`Grid`(key[, plotter, index_in_list, ...])	Display the grid
`Hist2DXRange`(*args, **kwargs)	Specify the range of the histogram for the x-dimension
`Hist2DYRange`(*args, **kwargs)	Specify the range of the histogram for the x-dimension

Continued on next page

Table 37 – continued from previous page

`HistBins`(key[, plotter, index_in_list, . . .])	Specify the bins of the 2D-Histogramm
`InterpolateBounds`(key[, plotter, . . .])	Interpolate grid cell boundaries for 2D plots
`LabelOptions`(key[, plotter, index_in_list, . . .])	Base formatoption class for label sizes
`LabelProps`(key[, plotter, index_in_list, . . .])	Set the font properties of both, x- and y-label
`LabelSize`(key[, plotter, index_in_list, . . .])	Set the size of both, x- and y-label
`LabelWeight`(key[, plotter, index_in_list, . . .])	Set the font size of both, x- and y-label
`Legend`(key[, plotter, index_in_list, . . .])	Draw a legend
`LegendLabels`(key[, plotter, index_in_list, . . .])	Set the labels of the arrays in the legend
`LimitBase`(*args, **kwargs)	Base class for x- and y-limits
`LineColors`(*args, **kwargs)	Set the color coding
`LinePlot`(*args, **kwargs)	Choose the line style of the plot
`LineWidth`(key[, plotter, index_in_list, . . .])	Choose the width of the lines
`Marker`(key[, plotter, index_in_list, . . .])	Choose the marker for points
`MarkerSize`(key[, plotter, index_in_list, . . .])	Choose the size of the markers for points
`MeanCalculator`(key[, plotter, . . .])	Determine how the error is visualized
`MissColor`(key[, plotter, index_in_list, . . .])	Set the color for missing values
`NormedHist2D`(key[, plotter, index_in_list, . . .])	Specify the normalization of the histogram
`Plot2D`(*args, **kwargs)	Choose how to visualize a 2-dimensional scalar data field
`PointDensity`(key[, plotter, index_in_list, . . .])	Specify the method to calculate the density
`SimplePlot2D`(*args, **kwargs)	Specify the plotting method
`SimpleVectorPlot`(*args, **kwargs)	Choose the vector plot type
`SymmetricLimits`(key[, plotter, . . .])	Make x- and y-axis symmetric
`TickLabels`(*args, **kwargs)	
`TickLabelsBase`(*args, **kwargs)	Abstract base class for ticklabels
`TickPropsBase`(key[, plotter, index_in_list, . . .])	Abstract base class for tick parameters
`TickSize`(key[, plotter, index_in_list, . . .])	Change the ticksize of the ticklabels
`TickSizeBase`(key[, plotter, index_in_list, . . .])	Abstract base class for modifying tick sizes
`TickWeight`(key[, plotter, index_in_list, . . .])	Change the fontweight of the ticks
`TickWeightBase`(key[, plotter, . . .])	Abstract base class for modifying font weight of ticks
`TicksBase`(*args, **kwargs)	Abstract base class for calculating ticks
`TicksManager`(key[, plotter, index_in_list, . . .])	Abstract base class for ticks formatoptions controlling major and minor
`TicksManagerBase`(key[, plotter, . . .])	Abstract base class for formatoptions handling ticks
`TicksOptions`(key[, plotter, index_in_list, . . .])	Base class for ticklabels options that apply for x- and y-axis
`Transpose`(*args, **kwargs)	Switch x- and y-axes
`VCLabel`(key[, plotter, index_in_list, . . .])	Show the colorbar label of the vector plot
`VectorBounds`(*args, **kwargs)	Specify the boundaries of the vector colorbar
`VectorCTicks`(*args, **kwargs)	Specify the tick locations of the vector colorbar
`VectorCalculator`(*args, **kwargs)	Abstract formatoption that provides calculation functions for speed, etc.
`VectorCbar`(*args, **kwargs)	Specify the position of the vector plot colorbars
`VectorColor`(*args, **kwargs)	Set the color for the arrows
`VectorLineWidth`(*args, **kwargs)	Change the linewidth of the arrows
`VectorPlot`(*args, **kwargs)	Choose the vector plot type
`ViolinPlot`(*args, **kwargs)	Choose how to make the violin plot
`ViolinXTickLabels`(*args, **kwargs)	Modify the x-axis ticklabels
`ViolinXTicks`(*args, **kwargs)	Modify the x-axis ticks
`ViolinXlim`(*args, **kwargs)	Set the x-axis limits
`ViolinYTickLabels`(*args, **kwargs)	Modify the x-axis ticklabels
`ViolinYTicks`(*args, **kwargs)	Modify the y-axis ticks

Continued on next page

Table 37 – continued from previous page

ViolinYlim(*args, **kwargs)	Set the y-axis limits
XRotation(key[, plotter, index_in_list, . . .])	Rotate the x-axis ticks
XTickLabels(*args, **kwargs)	Modify the x-axis ticklabels
XTickProps(key[, plotter, index_in_list, . . .])	Specify the x-axis tick parameters
XTicks(*args, **kwargs)	Modify the x-axis ticks
XTicks2D(*args, **kwargs)	Modify the x-axis ticks
Xlabel(key[, plotter, index_in_list, . . .])	Set the x-axis label
Xlim(*args, **kwargs)	Set the x-axis limits
Xlim2D(*args, **kwargs)	Set the x-axis limits
YRotation(key[, plotter, index_in_list, . . .])	Rotate the y-axis ticks
YTickLabels(*args, **kwargs)	Modify the y-axis ticklabels
YTickProps(key[, plotter, index_in_list, . . .])	Specify the y-axis tick parameters
YTicks(*args, **kwargs)	Modify the y-axis ticks
YTicks2D(*args, **kwargs)	Modify the y-axis ticks
Ylabel(key[, plotter, index_in_list, . . .])	Set the y-axis label
Ylim(*args, **kwargs)	Set the y-axis limits
Ylim2D(*args, **kwargs)	Set the y-axis limits

Plotter classes

BarPlotter([data, ax, auto_update, project, . . .])	Plotter for making bar plots
Base2D([data, ax, auto_update, project, . . .])	Base plotter for 2-dimensional plots
BaseVectorPlotter([data, ax, auto_update, . . .])	Base plotter for vector plots
CombinedBase([data, ax, auto_update, . . .])	Base plotter for combined 2-dimensional scalar and vector plot
CombinedSimplePlotter([data, ax, . . .])	Combined 2D plotter and vector plotter
DensityPlotter([data, ax, auto_update, . . .])	A plotter to visualize the density of points in a 2-dimensional grid
FldmeanPlotter([data, ax, auto_update, . . .])	**param data** Data object that shall be visualized. If given and *plot* is True,
LinePlotter([data, ax, auto_update, . . .])	Plotter for simple one-dimensional line plots
ScalarCombinedBase([data, ax, auto_update, . . .])	Base plotter for combined 2-dimensional scalar field with any other
Simple2DBase([data, ax, auto_update, . . .])	Base class for Simple2DPlotter and
Simple2DPlotter([data, ax, auto_update, . . .])	Plotter for visualizing 2-dimensional data.
SimplePlotterBase([data, ax, auto_update, . . .])	Base class for all simple plotters
SimpleVectorPlotter([data, ax, auto_update, . . .])	Plotter for visualizing 2-dimensional vector data
ViolinPlotter([data, ax, auto_update, . . .])	Plotter for making violin plots
XYTickPlotter([data, ax, auto_update, . . .])	Plotter class for x- and y-ticks and x- and y- ticklabels

Functions

format_coord_func(ax, ref)	Create a function that can replace the
round_to_05(n[, exp, mode])	Round to the next 0.5-value.

class psy_simple.plotters.**AlternativeXCoord**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: psyplot.plotter.Formatoption

Use an alternative variable as x-coordinate

This formatoption let's you specify another variable in the base dataset of the data array in case you want to use this as the x-coordinate instead of the raw data

Possible types

Attributes

data_dependent	bool(x) -> bool
data_iterator	
group	str(object='') -> str
name	str(object='') -> str
priority	int(x=0) -> integer
use_raw_data	Bool.

Methods

diff(value)	Checks whether the given value differs from what is currently set
get_alternative_coord(da, i)	
replace_coord(i)	Replace the coordinate for the data array at the given position
update(value)	Method that is call to update the formatoption on the axes

- *None* – Use the default
- *str* – The name of the variable to use in the base dataset
- *xarray.DataArray* – An alternative variable with the same shape as the displayed array

Examples

To see the difference, we create a simple test dataset:

```
>>> import xarray as xr

>>> import numpy as np

>>> import psyplot.project as psy

>>> ds = xr.Dataset({
...     'temp': xr.Variable(('time', ), np.arange(5)),
...     'std': xr.Variable(('time', ), np.arange(5, 10))})
>>> ds
<xarray.Dataset>
Dimensions:  (time: 5)
Coordinates:
  * time     (time) int64 0 1 2 3 4
Data variables:
    temp     (time) int64 0 1 2 3 4
    std      (time) int64 5 6 7 8 9
```

If we create a plot with it, we get the 'time' dimension on the x-axis:

```
>>> plotter = psy.plot.lineplot(ds, name=['temp']).plotters[0]

>>> plotter.plot_data[0].dims
('time',)
```

If we however set the 'coord' keyword, we get:

```
>>> plotter = psy.plot.lineplot(
...     ds, name=['temp'], coord='std').plotters[0]

>>> plotter.plot_data[0].dims
('std',)
```

and 'std' is plotted on the x-axis.

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the dependencies attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the children, dependencies and connections attributes, with values being the name of the new formatoption in this plotter.

data_dependent = True

data_iterator

diff(*value*)

Checks whether the given value differs from what is currently set

Parameters value – A possible value to set (make sure that it has been validate via the validate attribute before)

Returns True if the value differs from what is currently set

Return type bool

get_alternative_coord(*da, i*)

group = 'data'

name = 'Alternative X-Variable'

priority = 30

replace_coord(*i*)

Replace the coordinate for the data array at the given position

Parameters

- **i** (*int*) – The number of the data array in the raw data (if the raw data is not an interactive list, use 0)
- **Returns** –
- **xarray.DataArray** – The data array with the replaced coordinate

update(*value*)

Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

use_raw_data = True

Bool. If True, this Formatoption directly uses the raw_data, otherwise use the normal data

class psy_simple.plotters.**AlternativeXCoordPost**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.AlternativeXCoord*

Use an alternative variable as x-coordinate

This formatoption let's you specify another variable in the base dataset of the data array in case you want to use this as the x-coordinate instead of the raw data

Possible types

Attributes

use_raw_data	bool(x) -> bool

- *None* – Use the default
- *str* – The name of the variable to use in the base dataset
- *xarray.DataArray* – An alternative variable with the same shape as the displayed array

Examples

To see the difference, we create a simple test dataset:

```
>>> import xarray as xr

>>> import numpy as np

>>> import psyplot.project as psy

>>> ds = xr.Dataset({
...     'temp': xr.Variable(('time', ), np.arange(5)),
...     'std': xr.Variable(('time', ), np.arange(5, 10))})
>>> ds
<xarray.Dataset>
Dimensions:  (time: 5)
Coordinates:
  * time     (time) int64 0 1 2 3 4
```

(continues on next page)

(continued from previous page)

```
Data variables:
    temp     (time) int64 0 1 2 3 4
    std      (time) int64 5 6 7 8 9
```

If we create a plot with it, we get the `'time'` dimension on the x-axis:

```
>>> plotter = psy.plot.lineplot(ds, name=['temp']).plotters[0]

>>> plotter.plot_data[0].dims
('time',)
```

If we however set the `'coord'` keyword, we get:

```
>>> plotter = psy.plot.lineplot(
...     ds, name=['temp'], coord='std').plotters[0]

>>> plotter.plot_data[0].dims
('std',)
```

and `'std'` is plotted on the x-axis.

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

use_raw_data = False

class psy_simple.plotters.**ArrowSize**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Change the size of the arrows

Possible types

Attributes

dependencies	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
priority	int(x=0) -> integer

Methods

update(value)	Method that is call to update the formatoption on the axes

- *None* – make no scaling
- *float* – Factor scaling the size of the arrows

See also:

`arrowstyle`, `linewidth`, `density`, `color`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, *dependencies* and `connections` attributes, with values being the name of the new formatoption in this plotter.

dependencies = ['plot']

group = 'vector'

name = 'Size of the arrows'

plot
plot Formatoption instance in the plotter

priority = 20

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**ArrowStyle**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Change the style of the arrows

Possible types

Attributes

dependencies	list() -> new empty list
group	str(object=' ') -> str
name	str(object=' ') -> str
plot	plot Formatoption instance in the plotter
priority	int(x=0) -> integer

Methods

update(value)	Method that is call to update the formatoption on the axes

str – Any arrow style string (see `FancyArrowPatch`)

Notes

This formatoption only has an effect for stream plots

See also:

`arrowsize`, `linewidth`, `density`, `color`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, *dependencies* and `connections` attributes, with values being the name of the new formatoption in this plotter.

dependencies = ['plot']

group = 'vector'

name = 'Style of the arrows'

plot

plot Formatoption instance in the plotter

priority = 20

update(*value*)

Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**AxisColor**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.DictFormatoption`

Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

Attributes

`group`	str(object=' ') -> str
`name`	str(object=' ') -> str
`value2pickle`	Return the current axis colors

Methods

`initialize_plot`(value)	Method that is called when the plot is made the first time
`update`(value)	Method that is call to update the formatoption on the axes

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

Parameters

- **key** (`str`) – formatoption key in the *plotter*

- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

group = 'axes'

initialize_plot(*value*)

Method that is called when the plot is made the first time

Parameters **value** – The value to use for the initialization

name = 'Color of x- and y-axes'

update(*value*)

Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

value2pickle

Return the current axis colors

class psy_simple.plotters.**BarAlpha**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Specify the transparency (alpha)

Possible types

Attributes

priority	int(x=0) -> integer

Methods

update(value)	Method that is call to update the formatoption on the axes

float – A value between 0 (opaque) and 1 invisible

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.

- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

priority = 20

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**BarPlot**(**args*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Choose how to make the bar plot

Possible types

Attributes

alpha	alpha Formatoption instance in the plotter
children	list() -> new empty list
color	color Formatoption instance in the plotter
data_dependent	bool(x) -> bool
dependencies	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
plot_fmt	bool(x) -> bool
priority	int(x=0) -> integer
transpose	transpose Formatoption instance in the plotter
widths	widths Formatoption instance in the plotter

Methods

get_xys(arr)	
make_plot()	
remove()	Method to remove the effects of this formatoption
update(value)	Method that is call to update the formatoption on the axes

- *None* – Don't make any plotting
- *'bar'* – Create a usual bar plot with the bars side-by-side
- *'stacked'* – Create stacked plot

alpha
alpha Formatoption instance in the plotter

children = ['color', 'transpose', 'alpha']

color
color Formatoption instance in the plotter

data_dependent = True

dependencies = ['widths']

get_xys(*arr*)

group = 'plotting'

make_plot()

name = 'Bar plot type'

plot_fmt = True

priority = 20

remove()
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

transpose
transpose Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

widths
widths Formatoption instance in the plotter

class psy_simple.plotters.**BarPlotter**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)
Bases: `psy_simple.plotters.SimplePlotterBase`

Plotter for making bar plots

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (`matplotlib.axes.Axes`) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary

- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the `post` formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Miscallaneous formatoptions

`alpha`	Specify the transparency (alpha)
`widths`	Specify the widths of the bars
`legend`	Draw a legend
`legendlabels`	Set the labels of the arrays in the legend
`sym_lims`	Make x- and y-axis symmetric
`ticksize`	Change the ticksize of the ticklabels
`tickweight`	Change the fontweight of the ticks

Data manipulation formatoptions

`coord`	Use an alternative variable as x-coordinate

Plot formatoptions

`plot`	Choose how to make the bar plot

Label formatoptions

`xlabel`	Set the x-axis label
`ylabel`	Set the y-axis label
`figtitle`	Plot a figure title
`figtitleprops`	Properties of the figure title
`figtitlesize`	Set the size of the figure title
`figtitleweight`	Set the fontweight of the figure title
`labelprops`	Set the font properties of both, x- and y-label
`labelsize`	Set the size of both, x- and y-label
`labelweight`	Set the font size of both, x- and y-label
`text`	Add text anywhere on the plot
`title`	Show the title
`titleprops`	Properties of the title
`titlesize`	Set the size of the title
`titleweight`	Set the fontweight of the title

Axes formatoptions

`xlim`	Set the x-axis limits
`ylim`	Set the y-axis limits
`axiscolor`	Color the x- and y-axes
`grid`	Display the grid
`tight`	Automatically adjust the plots.

Continued on next page

Table 57 – continued from previous page

transpose	Switch x- and y-axes

Axis tick formatoptions

xticklabels	Modify the x-axis ticklabels
yticklabels	Modify the y-axis ticklabels
xrotation	Rotate the x-axis ticks
xtickprops	Specify the x-axis tick parameters
xticks	Modify the x-axis ticks
yrotation	Rotate the y-axis ticks
ytickprops	Specify the y-axis tick parameters
yticks	Modify the y-axis ticks

Color coding formatoptions

color	Set the color coding

Masking formatoptions

maskbetween	Mask data points between two numbers
maskgeq	Mask data points greater than or equal to a number
maskgreater	Mask data points greater than a number
maskleq	Mask data points smaller than or equal to a number
maskless	Mask data points smaller than a number

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

alpha

Specify the transparency (alpha)

Possible types

float – A value between 0 (opaque) and 1 invisible

coord

Use an alternative variable as x-coordinate

This formatoption let's you specify another variable in the base dataset of the data array in case you want to use this as the x-coordinate instead of the raw data

Possible types

- *None* – Use the default
- *str* – The name of the variable to use in the base dataset
- *xarray.DataArray* – An alternative variable with the same shape as the displayed array

Examples

To see the difference, we create a simple test dataset:

```
>>> import xarray as xr

>>> import numpy as np

>>> import psyplot.project as psy

>>> ds = xr.Dataset({
...     'temp': xr.Variable(('time', ), np.arange(5)),
...     'std': xr.Variable(('time', ), np.arange(5, 10))})
>>> ds
<xarray.Dataset>
Dimensions:  (time: 5)
Coordinates:
  * time     (time) int64 0 1 2 3 4
Data variables:
    temp     (time) int64 0 1 2 3 4
    std      (time) int64 5 6 7 8 9
```

If we create a plot with it, we get the `'time'` dimension on the x-axis:

```
>>> plotter = psy.plot.lineplot(ds, name=['temp']).plotters[0]

>>> plotter.plot_data[0].dims
('time',)
```

If we however set the `'coord'` keyword, we get:

```
>>> plotter = psy.plot.lineplot(
...     ds, name=['temp'], coord='std').plotters[0]

>>> plotter.plot_data[0].dims
('std',)
```

and `'std'` is plotted on the x-axis.

plot
Choose how to make the bar plot

Possible types

- *None* – Don't make any plotting
- *'bar'* – Create a usual bar plot with the bars side-by-side
- *'stacked'* – Create stacked plot

widths
Specify the widths of the bars

Possible types

- *'equal'* – Each bar will have the same width (the default)
- *'data'* – Each bar will have the width as specified by the boundaries

`xlabel`
Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

`xlim`
Set the x-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

minmax Uses the minimum and maximum

sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

xticklabels
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

ylabel
Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

ylim
Set the y-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`xlim`

yticklabels
Modify the y-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`yticks`, *`ticksize`*, *`tickweight`*, *`ytickprops`*, *`xticklabels`*

color

Set the color coding

This formatoptions sets the color of the lines, bars, etc.

Possible types

- *None* – to use the axes color_cycle
- *iterable* – (e.g. list) to specify the colors manually
- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.ColorMap* – to automatically choose the colors according to the number of lines, etc. from the given colormap

maskbetween

Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

`maskless`, *`maskleq`*, *`maskgreater`*, *`maskgeq`*

maskgeq

Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

`maskless`, *`maskleq`*, *`maskgreater`*, *`maskbetween`*

maskgreater

Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

`maskless`, *`maskleq`*, *`maskgeq`*, *`maskbetween`*

maskleq

Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

figtitle
Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this `Figtitle` instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *title* formatoption.

See also:

title, *figtitlesize*, *figtitleweight*, *figtitleprops*

figtitleprops
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

figtitle, *figtitlesize*, *figtitleweight*

figtitlesize
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

figtitle, *figtitleweight*, *figtitleprops*

figtitleweight
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

figtitle, *figtitlesize*, *figtitleprops*

labelprops
Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

xlabel, *ylabel*, *labelsize*, *labelweight*

labelsize
Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

xlabel, *ylabel*, *labelweight*, *labelprops*

labelweight
Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

xlabel, *ylabel*, *labelsize*, *labelprops*

text
Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`

- desc: `%(long_name)s [%(units)s]`
- dinfo: `%B %d, %Y`
- tinfo: `%H:%M`
- sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

`title`, *`figtitle`*

`title`

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the `figtitle` formatoption.

See also:

`figtitle`, `titlesize`, `titleweight`, `titleprops`

titleprops
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`title`, `titlesize`, `titleweight`

titlesize
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`title`, `titleweight`, `titleprops`

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`title`, `titlesize`, `titleprops`

post
Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

axiscolor

Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

grid

Display the grid

Show the grid on the plot with the specified color.

Possible types

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

tight
Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

> **Warning:** There is no update method to undo what happend after this formatoption is set to True!

transpose
Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

xrotation
Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

yrotation

xtickprops
Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *ytickprops*

xticks
Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

`xticklabels`, `ticksize`, `tickweight`, `xtickprops`, `yticks`

yrotation

Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`xrotation`

ytickprops

Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`xticks`, `yticks`, `ticksize`, `tickweight`, `xtickprops`

yticks

Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

`yticklabels`, `ticksize`, `tickweight`, `ytickprops`

`xticks` for possible examples

`legend`

Draw a legend

This formatoption determines where and if to draw the legend. It uses the `labels` formatoption to determine the labels.

Possible types

- *bool* – Draw a legend or not
- *str or int* – Specifies where to plot the legend (i.e. the location)
- *dict* – Give the keywords for the `matplotlib.pyplot.legend()` function

See also:

`labels`

`legendlabels`

Set the labels of the arrays in the legend

This formatoption specifies the labels for each array in the legend. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – A single string that shall be used for all arrays.
- *list of str* – Same as a single string but specified for each array

See also:

`legend`

`sym_lims`

Make x- and y-axis symmetric

Possible types

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

ticksize
Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

tickweight, *xtickprops*, *ytickprops*

tickweight
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

ticksize, *xtickprops*, *ytickprops*

class psy_simple.plotters.**BarWidths**(*key, plotter=None, index_in_list=None, additional_children=[], additional_dependencies=[], **kwargs*)

Bases: `psyplot.plotter.Formatoption`

Specify the widths of the bars

Possible types

Attributes

priority	int(x=0) -> integer

Methods

update(value)	Method that is call to update the formatoption on the axes

- *'equal'* – Each bar will have the same width (the default)
- *'data'* – Each bar will have the width as specified by the boundaries

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

priority = 20

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**BarXTickLabels**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.XTickLabels*

Modify the x-axis ticklabels

Possible types

Attributes

dependencies	list() -> new empty list
plot	plot Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter
widths	widths Formatoption instance in the plotter
xticks	xticks Formatoption instance in the plotter
yticklabels	yticklabels Formatoption instance in the plotter

Methods

set_stringformatter(s)

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

xticks, `ticksize`, `tickweight`, `xtickprops`, *yticklabels*

dependencies = ['transpose', 'xticks', 'yticklabels', 'plot', 'widths']

plot
plot Formatoption instance in the plotter

set_stringformatter(*s*)

transpose
transpose Formatoption instance in the plotter

widths
widths Formatoption instance in the plotter

xticks
xticks Formatoption instance in the plotter

yticklabels
yticklabels Formatoption instance in the plotter

class psy_simple.plotters.**BarXlabel**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.Xlabel*

Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Attributes

transpose	transpose Formatoption instance in the plotter
update_after_plot	Xlabel is modified by the pandas plot routine, therefore we update it
ylabel	ylabel Formatoption instance in the plotter

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

transpose
transpose Formatoption instance in the plotter

update_after_plot = True
Xlabel is modified by the pandas plot routine, therefore we update it after each plot

ylabel
ylabel Formatoption instance in the plotter

class psy_simple.plotters.**BarXlim**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.ViolinXlim*

Set the x-axis limits

Possible types

Attributes

array	The numpy array of the data
dependencies	list() -> new empty list

Continued on next page

Table 67 – continued from previous page

`plot`	plot Formatoption instance in the plotter
`sym_lims`	sym_lims Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter
`widths`	widths Formatoption instance in the plotter
`xticks`	xticks Formatoption instance in the plotter
`ylim`	ylim Formatoption instance in the plotter

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

array
The numpy array of the data

dependencies = ['xticks', 'widths']

plot
plot Formatoption instance in the plotter

sym_lims
sym_lims Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

widths
widths Formatoption instance in the plotter

xticks
xticks Formatoption instance in the plotter

ylim
ylim Formatoption instance in the plotter

class psy_simple.plotters.**BarYTickLabels**(**args*, ***kwargs*)
Bases: `psy_simple.plotters.YTickLabels`

Modify the y-axis ticklabels

Possible types

Attributes

dependencies	list() -> new empty list
plot	plot Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter
yticks	yticks Formatoption instance in the plotter

Methods

set_stringformatter(s)

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

yticks, `ticksize`, `tickweight`, `ytickprops`, `xticklabels`

dependencies = ['transpose', 'yticks', 'plot']

plot
plot Formatoption instance in the plotter

set_stringformatter(*s*)

transpose
transpose Formatoption instance in the plotter

yticks
yticks Formatoption instance in the plotter

class psy_simple.plotters.**BarYlabel**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.Ylabel*

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).

- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Attributes

transpose	transpose Formatoption instance in the plotter
update_after_plot	Ylabel is modified by the pandas plot routine, therefore we update it

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

transpose
transpose Formatoption instance in the plotter

update_after_plot = True
Ylabel is modified by the pandas plot routine, therefore we update it after each plot

class psy_simple.plotters.**BarYlim**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.ViolinYlim*

Set the y-axis limits

Possible types

Attributes

`array`	The numpy array of the data
`dependencies`	list() -> new empty list
`plot`	plot Formatoption instance in the plotter
`sym_lims`	sym_lims Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter
`widths`	widths Formatoption instance in the plotter
`xlim`	xlim Formatoption instance in the plotter
`yticks`	yticks Formatoption instance in the plotter

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`xlim`

`array`
The numpy array of the data

`dependencies = ['yticks', 'widths']`

`plot`
plot Formatoption instance in the plotter

`sym_lims`
sym_lims Formatoption instance in the plotter

`transpose`
transpose Formatoption instance in the plotter

`widths`
widths Formatoption instance in the plotter

`xlim`
xlim Formatoption instance in the plotter

`yticks`
yticks Formatoption instance in the plotter

class psy_simple.plotters.**Base2D**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: psyplot.plotter.Plotter

Base plotter for 2-dimensional plots

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the initialize_plot() method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the initialize_plot() method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the update() method or not. See also the no_auto_update attribute. If None, the value from the 'lists.auto_update' key in the psyplot.rcParams dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the psyplot.rcParams dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the post formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the formatoptions attribute that shall be used

Color coding formatoptions

bounds	Specify the boundaries of the colorbar
cbar	Specify the position of the colorbars
cbarspacing	Specify the spacing of the bounds in the colorbar
cmap	Specify the color map
ctickprops	Specify the font properties of the colorbar ticklabels
cticksize	Specify the font size of the colorbar ticklabels
ctickweight	Specify the fontweight of the colorbar ticklabels
extend	Draw arrows at the side of the colorbar

Label formatoptions

clabel	Show the colorbar label
clabelprops	Properties of the Colorbar label
clabelsize	Set the size of the Colorbar label
clabelweight	Set the fontweight of the Colorbar label

Attributes

convert_radian	Boolean that is True if triangles with units in radian should be

Axis tick formatoptions

cticklabels	Specify the colorbar ticklabels
cticks	Specify the tick locations of the colorbar

Miscallaneous formatoptions

datagrid	Show the grid of the data

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

bounds

Specify the boundaries of the colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

cmap Specifies the colormap

cbar

Specify the position of the colorbars

Possible types

- *bool* – True: defaults to 'b' False: Don't draw any colorbar
- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

Examples

Draw a colorbar at the bottom and left of the axes:

```
>>> plotter.update(cbar='bl')
```

---

cbarspacing

Specify the spacing of the bounds in the colorbar

Possible types

str {'uniform', 'proportional'} – if `'uniform'`, every color has exactly the same width in the colorbar, if `'proportional'`, the size is chosen according to the data

clabel

Show the colorbar label

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)

- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `set_label()` method.

See also:

`clabelsize`, `clabelweight`, `clabelprops`

clabelprops
Properties of the Colorbar label

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`clabel`, `clabelsize`, `clabelweight`

clabelsize
Set the size of the Colorbar label

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`clabel`, `clabelweight`, `clabelprops`

clabelweight
Set the fontweight of the Colorbar label

Possible types

- *float* – a float between 0 and 1000

- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`clabel`, `clabelsize`, `clabelprops`

cmap
Specify the color map

This formatoption specifies the color coding of the data via a `matplotlib.colors.Colormap`

Possible types

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

`bounds` specifies the boundaries of the colormap

convert_radian = False
Boolean that is True if triangles with units in radian should be converted to degrees

cticklabels
Specify the colorbar ticklabels

Possible types

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`cticks`, `cticksize`, `ctickweight`, `ctickprops`, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

ctickprops
Specify the font properties of the colorbar ticklabels

Possible types

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`cticksize`, `ctickweight`, `cticklabels`, `cticks`, `vcticksize`, `vctickweight`, `vcticklabels`, `vcticks`

cticks
Specify the tick locations of the colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, …]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  bounds let the `bounds` keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`

`cticksize`
Specify the font size of the colorbar ticklabels

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`ctickweight`, `ctickprops`, `cticklabels`, `cticks`, `vctickweight`, `vctickprops`, `vcticklabels`, `vcticks`

`ctickweight`
Specify the fontweight of the colorbar ticklabels

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

ctickssize, *ctickprops*, *cticklabels*, *cticks*, vcticksize, vctickprops, vcticklabels, vcticks

datagrid

Show the grid of the data

This formatoption shows the grid of the data (without labels)

Possible types

- *None* – Don't show the data grid
- *str* – A linestyle in the form `'k-'`, where `'k'` is the color and `'-'` the linestyle.
- *dict* – any keyword arguments that are passed to the plotting function (`matplotlib.pyplot.triplot()` for triangular grids and `matplotlib.pyplot.hlines()` for rectilinear grids)

extend

Draw arrows at the side of the colorbar

Possible types

str {'neither', 'both', 'min' or 'max'} – If not 'neither', make pointed end(s) for out-of-range values

plot = None

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

class psy_simple.plotters.**BaseVectorPlotter**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: *psy_simple.plotters.Base2D*

Base plotter for vector plots

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also

the no_auto_update attribute. If None, the value from the 'lists.auto_update' key in the psyplot.rcParams dictionary is used.

- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the psyplot.rcParams dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the *post* formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the formatoptions attribute that shall be used

Vector plot formatoptions

arrowsize	Change the size of the arrows
arrowstyle	Change the style of the arrows
density	Change the density of the arrows

Color coding formatoptions

bounds	Specify the boundaries of the vector colorbar
cbar	Specify the position of the vector plot colorbars
color	Set the color for the arrows
cbarspacing	Specify the spacing of the bounds in the colorbar
cmap	Specify the color map
ctickprops	Specify the font properties of the colorbar ticklabels
cticksize	Specify the font size of the colorbar ticklabels
ctickweight	Specify the fontweight of the colorbar ticklabels
extend	Draw arrows at the side of the colorbar

Methods

check_data(name, dims, is_unstructured)	A validation method for the data shape

Axis tick formatoptions

cticks	Specify the tick locations of the vector colorbar
cticklabels	Specify the colorbar ticklabels

Miscallaneous formatoptions

linewidth	Change the linewidth of the arrows
datagrid	Show the grid of the data

Label formatoptions

clabel	Show the colorbar label

Continued on next page

Table 83 – continued from previous page

clabelprops	Properties of the Colorbar label
clabelsize	Set the size of the Colorbar label
clabelweight	Set the fontweight of the Colorbar label

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

arrowsize
Change the size of the arrows

Possible types

- *None* – make no scaling
- *float* – Factor scaling the size of the arrows

See also:

arrowstyle, *linewidth*, *density*, *color*

arrowstyle
Change the style of the arrows

Possible types

str – Any arrow style string (see `FancyArrowPatch`)

Notes

This formatoption only has an effect for stream plots

See also:

arrowsize, *linewidth*, *density*, *color*

bounds
Specify the boundaries of the vector colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

cmap Specifies the colormap

cbar
Specify the position of the vector plot colorbars

Possible types

- *bool* – True: defaults to 'b' False: Don't draw any colorbar
- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

classmethod check_data(*name*, *dims*, *is_unstructured*)
A validation method for the data shape

Parameters

- **name** (*str or list of str*) – The variable names (two variables for the array or one if the dims are one greater)
- **dims** (*list with length 1 or list of lists with length 1*) – The dimension of the arrays. Only 2D-Arrays are allowed (or 1-D if the array is unstructured)
- **is_unstructured** (*bool or list of bool*) – True if the corresponding array is unstructured.

Returns

- *list of bool or None* – True, if everything is okay, False in case of a serious error, None if it is intermediate. Each object in this list corresponds to one in the given *name*
- *list of str* – The message giving more information on the reason. Each object in this list corresponds to one in the given *name*

color

Set the color for the arrows

This formatoption can be used to set a single color for the vectors or define the color coding

Possible types

- *float* – Determines the greyness
- *color* – Defines the same color for all arrows. The string can be either a html hex string (e.g. '#eeefff'), a single letter (e.g. 'b': blue, 'g': green, 'r': red, 'c': cyan, 'm': magenta, 'y': yellow, 'k': black, 'w': white) or any other color
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *2D-array* – The values determine the color for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

`arrowsize`, `arrowstyle`, `density`, `linewidth`

cticks

Specify the tick locations of the vector colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

data plot the ticks exactly where the data is.

mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

bounds let the `bounds` keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`, `vcticklabels`

`density`
Change the density of the arrows

Possible types

- *float* – Scales the density of the arrows in x- and y-direction (1.0 means no scaling)
- *tuple (x, y)* – Defines the scaling in x- and y-direction manually

Notes

quiver plots do not support density scaling

`linewidth`
Change the linewidth of the arrows

Possible types

- *float* – give the linewidth explicitly
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *tuple (string, float)* – *string* may be one of the above strings, *float* may be a scaling factor
- *2D-array* – The values determine the linewidth for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

arrowsize, *arrowstyle*, *density*, *color*

cbarspacing

Specify the spacing of the bounds in the colorbar

Possible types

str {'uniform', 'proportional'} – if 'uniform', every color has exactly the same width in the colorbar, if 'proportional', the size is chosen according to the data

cmap

Specify the color map

This formatoption specifies the color coding of the data via a matplotlib.colors.Colormap

Possible types

- *str* – Strings may be any valid colormap name suitable for the matplotlib.cm.get_cmap() function or one of the color lists defined in the 'colors.cmaps' key of the psyplot.rcParams dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

bounds specifies the boundaries of the colormap

ctickprops

Specify the font properties of the colorbar ticklabels

Possible types

dict – Items may be anything of the matplotlib.pyplot.tick_params() function

See also:

cticksize, *ctickweight*, *cticklabels*, *cticks*, vcticksize, vctickweight, vcticklabels, vcticks

cticksize

Specify the font size of the colorbar ticklabels

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

ctickweight, *ctickprops*, *cticklabels*, *cticks*, vctickweight, vctickprops, vcticklabels, vcticks

ctickweight

Specify the fontweight of the colorbar ticklabels

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

cticksize, *ctickprops*, *cticklabels*, *cticks*, vcticksize, vctickprops, vcticklabels, vcticks

extend

Draw arrows at the side of the colorbar

Possible types

str {'neither', 'both', 'min' or 'max'} – If not 'neither', make pointed end(s) for out-of-range values

clabel

Show the colorbar label

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `set_label()` method.

See also:

clabelsize, *clabelweight*, *clabelprops*

clabelprops
Properties of the Colorbar label

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`clabel`, `clabelsize`, `clabelweight`

clabelsize
Set the size of the Colorbar label

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`clabel`, `clabelweight`, `clabelprops`

clabelweight
Set the fontweight of the Colorbar label

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`clabel`, `clabelsize`, `clabelprops`

post
Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post`

attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing
Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

cticklabels
Specify the colorbar ticklabels

Possible types

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers

- *array* – An array of strings to use for the ticklabels

See also:

`cticks`, `cticksize`, `ctickweight`, `ctickprops`, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

datagrid
Show the grid of the data

This formatoption shows the grid of the data (without labels)

Possible types

- *None* – Don't show the data grid
- *str* – A linestyle in the form `'k-'`, where `'k'` is the color and `'-'` the linestyle.
- *dict* – any keyword arguments that are passed to the plotting function (`matplotlib.pyplot.triplot()` for triangular grids and `matplotlib.pyplot.hlines()` for rectilinear grids)

class `psy_simple.plotters.`**Bounds**(**args*, ***kwargs*)
Bases: `psy_simple.plotters.DataTicksCalculator`

Specify the boundaries of the colorbar

Possible types

Attributes

`cbar`	cbar Formatoption instance in the plotter
`cmap`	cmap Formatoption instance in the plotter
`connections`	list() -> new empty list
`group`	str(object='') -> str
`name`	str(object='') -> str
`priority`	int(x=0) -> integer
`value2share`	The normalization instance

Methods

`get_fmt_widget`(parent, project)	Open a `psy_simple.widget.CMapFmtWidget`
`update`(value)	Method that is call to update the formatoption on the axes

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, …]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

cmap Specifies the colormap

`cbar`
cbar Formatoption instance in the plotter

`cmap`
cmap Formatoption instance in the plotter

`connections = ['cmap', 'cbar']`

`get_fmt_widget` (*parent*, *project*)
Open a `psy_simple.widget.CMapFmtWidget`

`group = 'colors'`

`name = 'Boundaries of the color map'`

`priority = 20`

`update` (*value*)
Method that is call to update the formatoption on the axes

Parameters **`value`** – Value to update

`value2share`
The normalization instance

class psy_simple.plotters.**CLabel**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psy_simple.base.TextBase`, `psyplot.plotter.Formatoption`

Show the colorbar label

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Attributes

`axis_locations`	dict() -> new empty dictionary
`cbar`	cbar Formatoption instance in the plotter
`children`	list() -> new empty list
`data_dependent`	bool(x) -> bool
`dependencies`	list() -> new empty list
`group`	str(object='') -> str
`name`	str(object='') -> str
`plot`	plot Formatoption instance in the plotter

Methods

`update`(value)	Method that is call to update the formatoption on the axes

Possible types

str – The title for the `set_label()` method.

See also:

`clabelsize`, `clabelweight`, `clabelprops`

Parameters

- **key** (*str*) – formatoption key in the *plotter*

- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the *children* attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the the *children*, *dependencies* and `connections` attributes, with values being the name of the new formatoption in this plotter.

axis_locations = {'b': 'x', 'fb': 'x', 'fl': 'y', 'fr': 'y', 'ft': 'x', 'l': 'y'

cbar
cbar Formatoption instance in the plotter

children = ['plot']

data_dependent = True

dependencies = ['cbar']

group = 'labels'

name = 'Colorbar label'

plot
plot Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**CMap**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Specify the color map

This formatoption specifies the color coding of the data via a `matplotlib.colors.Colormap`

Possible types

Attributes

bounds	bounds Formatoption instance in the plotter
cbar	cbar Formatoption instance in the plotter
connections	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
priority	int(x=0) -> integer

Methods

get_cmap([arr, cmap, N])	Get the `matplotlib.colors.Colormap` for plotting
get_fmt_widget(parent, project)	Open a `psy_simple.widget.CMapFmtWidget`
update(value)	Method that is call to update the formatoption on the axes

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

bounds specifies the boundaries of the colormap

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and *connections* attributes, with values being the name of the new formatoption in this plotter.

bounds
bounds Formatoption instance in the plotter

cbar
cbar Formatoption instance in the plotter

connections = ['bounds', 'cbar']

get_cmap(*arr=None*, *cmap=None*, *N=None*)
Get the `matplotlib.colors.Colormap` for plotting

Parameters

- **arr** (*np.ndarray*) – The array to plot
- **cmap** (*str or matplotlib.colors.Colormap*) – The colormap to use. If None, the `value` of this formatoption is used
- **N** (*int*) – The number of colors in the colormap. If None, the norm of the *bounds* formatoption is used and, if necessary, the given array *arr*

Returns The colormap returned by *psy_simple.colors.get_cmap()*

Return type matplotlib.colors.Colormap

get_fmt_widget (*parent*, *project*)
Open a `psy_simple.widget.CMapFmtWidget`

group = 'colors'

name = 'Colormap'

priority = 20

update (*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**CTickLabels** (**args*, ***kwargs*)
Bases: *psy_simple.plotters.CbarOptions*, *psy_simple.plotters.TickLabelsBase*

Specify the colorbar ticklabels

Possible types

Attributes

cbar	cbar Formatoption instance in the plotter
default_formatters	Default locator of the axis of the colorbars
name	str(object='') -> str
plot	plot Formatoption instance in the plotter

Methods

set_default_formatters()	Sets the default formatters that is used for updating to None
set_formatter(formatter)	Sets a given formatter

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`cticks`, `cticksize`, `ctickweight`, `ctickprops`, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

cbar
cbar Formatoption instance in the plotter

default_formatters
Default locator of the axis of the colorbars

name = 'Colorbar ticklabels'

plot
plot Formatoption instance in the plotter

set_default_formatters ()
Sets the default formatters that is used for updating to None

set_formatter (*formatter*)
Sets a given formatter

class psy_simple.plotters.**CTickProps** (*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: *psy_simple.plotters.CbarOptions*, *psy_simple.plotters.TickPropsBase*

Specify the font properties of the colorbar ticklabels

Possible types

Attributes

cbar	cbar Formatoption instance in the plotter
children	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
plot	plot Formatoption instance in the plotter

Methods

update_axis(value)

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`cticksize`, `ctickweight`, `cticklabels`, `cticks`, `vcticksize`, `vctickweight`, `vcticklabels`, `vcticks`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the *children* attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the *children*, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

cbar
cbar Formatoption instance in the plotter

children = ['plot']

group = 'colors'

name = 'Font properties of the colorbar ticklabels'

plot
plot Formatoption instance in the plotter

update_axis(*value*)

class psy_simple.plotters.**CTickSize**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.CbarOptions*, *psy_simple.plotters.TickSizeBase*

Specify the font size of the colorbar ticklabels

Possible types

Attributes

cbar	cbar Formatoption instance in the plotter
ctickprops	ctickprops Formatoption instance in the plotter
dependencies	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
plot	plot Formatoption instance in the plotter

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

ctickweight, *ctickprops*, cticklabels, cticks, vctickweight, vctickprops, vcticklabels, vcticks

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the children, *dependencies* and connections attributes, with values being the name of the new formatoption in this plotter.

cbar
cbar Formatoption instance in the plotter

ctickprops
ctickprops Formatoption instance in the plotter

dependencies = ['cbar', 'ctickprops']

group = 'colors'

name = 'Font size of the colorbar ticklabels'

plot
plot Formatoption instance in the plotter

class psy_simple.plotters.**CTickWeight**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.CbarOptions*, *psy_simple.plotters.TickWeightBase*

Specify the fontweight of the colorbar ticklabels

Possible types

Attributes

cbar	cbar Formatoption instance in the plotter
ctickprops	ctickprops Formatoption instance in the plotter
dependencies	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
plot	plot Formatoption instance in the plotter

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

cticksize, *ctickprops*, cticklabels, cticks, vcticksize, vctickprops, vcticklabels, vcticks

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords

may be anything of the `children`, *`dependencies`* and `connections` attributes, with values being the name of the new formatoption in this plotter.

cbar
cbar Formatoption instance in the plotter

ctickprops
ctickprops Formatoption instance in the plotter

dependencies = ['cbar', 'ctickprops']

group = 'colors'

name = 'Font weight of the colorbar ticklabels'

plot
plot Formatoption instance in the plotter

class psy_simple.plotters.**CTicks**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.CbarOptions*, *psy_simple.plotters.TicksBase*

Specify the tick locations of the colorbar

Possible types

Attributes

bounds	bounds Formatoption instance in the plotter
cbar	cbar Formatoption instance in the plotter
default_locator	Default locator of the axis of the colorbars
dependencies	list() -> new empty list
name	str(object='') -> str
plot	plot Formatoption instance in the plotter

Methods

set_default_locators(*args, **kwargs)	Sets the default locator that is used for updating to None or int
set_ticks(value)	
update(value)	Method that is call to update the formatoption on the axes
update_axis(value)	

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

bounds let the *bounds* keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`

bounds
bounds Formatoption instance in the plotter

cbar
cbar Formatoption instance in the plotter

default_locator
Default locator of the axis of the colorbars

dependencies = ['cbar', 'bounds']

name = 'Colorbar ticks'

plot
plot Formatoption instance in the plotter

set_default_locators(**args*, ***kwargs*)
Sets the default locator that is used for updating to None or int

Parameters which (*{None, 'minor', 'major'}*) – Specify which locator shall be set

set_ticks(*value*)

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

update_axis(*value*)

class psy_simple.plotters.**Cbar**(**args*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Specify the position of the colorbars

Possible types

Attributes

bounds	bounds Formatoption instance in the plotter
cbarspacing	cbarspacing Formatoption instance in the plotter
cmap	cmap Formatoption instance in the plotter

Continued on next page

Table 99 – continued from previous page

dependencies	list() -> new empty list
extend	extend Formatoption instance in the plotter
figure_positions	set() -> new empty set object
group	str(object='') -> str
init_kwargs	dict key word arguments that are passed to the
levels	levels Formatoption instance in the plotter
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
priority	float(x) -> floating point number
value2share	Those colorbar positions that are directly at the axes

Methods

draw_colorbar(pos)	
finish_update()	Finish the update, initialization and sharing process
initialize_plot(value)	Method that is called when the plot is made the first time
remove([positions])	Method to remove the effects of this formatoption
update(value)	Updates the colorbar
update_colorbar(pos)	

- *bool* – True: defaults to 'b' False: Don't draw any colorbar
- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

Examples

Draw a colorbar at the bottom and left of the axes:

```
>>> plotter.update(cbar='bl')
```

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)

- **`**kwargs`** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, *`dependencies`* and `connections` attributes, with values being the name of the new formatoption in this plotter.
- **`other_cbars`** (*`list of str`*) – List of other colorbar formatoption keys (necessary for a sufficient resizing of the axes)

`bounds`
bounds Formatoption instance in the plotter

`cbarspacing`
cbarspacing Formatoption instance in the plotter

`cmap`
cmap Formatoption instance in the plotter

`dependencies = ['plot', 'cmap', 'bounds', 'extend', 'cbarspacing', 'levels']`

`draw_colorbar`(*pos*)

`extend`
extend Formatoption instance in the plotter

`figure_positions = {'l', 'fb', 'ft', 'fr', 't', 'r', 'b', 'fl'}`

`finish_update`()
Finish the update, initialization and sharing process

This function is called at the end of the `Plotter.start_update()`, `Plotter.initialize_plot()` or the `Plotter.share()` methods.

`group = 'colors'`

`init_kwargs`
`dict` key word arguments that are passed to the initialization of a new instance when accessed from the descriptor

`initialize_plot`(*value*)
Method that is called when the plot is made the first time

Parameters **`value`** – The value to use for the initialization

`levels`
levels Formatoption instance in the plotter

`name = 'Position of the colorbar'`

`original_position = None`

`plot`
plot Formatoption instance in the plotter

`priority = 10.1`

`remove`(*positions='all'*)
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

`update`(*value*)
Updates the colorbar

Parameters

- **value** – The value to update (see possible types)
- **no_fig_cbars** – Does not update the colorbars that are not in the axes of this plot

update_colorbar(*pos*)

value2share
Those colorbar positions that are directly at the axes

class psy_simple.plotters.**CbarOptions**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: psyplot.plotter.Formatoption

Base class for colorbar formatoptions

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the *children* attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the *children*, *dependencies* and connections attributes, with values being the name of the new formatoption in this plotter.

Attributes

axis	axis of the colorbar with the ticks. Will be overwritten during
axis_locations	dict() -> new empty dictionary
axisname	
cbar	cbar Formatoption instance in the plotter
children	list() -> new empty list
colorbar	
data	The data that is plotted
dependencies	list() -> new empty list
plot	plot Formatoption instance in the plotter
which	str(object='') -> str

Methods

update(value)	Method that is call to update the formatoption on the axes

axis
axis of the colorbar with the ticks. Will be overwritten during update process.

axis_locations = {'b': 'x', 'fb': 'x', 'fl': 'y', 'fr': 'y', 'ft': 'x', 'l': 'y'

axisname

cbar
cbar Formatoption instance in the plotter

children = ['plot']

colorbar

data
The data that is plotted

dependencies = ['cbar']

plot
plot Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

which = 'major'

class psy_simple.plotters.**CbarSpacing**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Specify the spacing of the bounds in the colorbar

Possible types

Attributes

cbar	cbar Formatoption instance in the plotter
connections	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str

Methods

update(value)	Method that is call to update the formatoption on the axes

str {'uniform', 'proportional'} – if `'uniform'`, every color has exactly the same width in the colorbar, if `'proportional'`, the size is chosen according to the data

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`

- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

cbar
cbar Formatoption instance in the plotter

connections = ['cbar']

group = 'colors'

name = 'Spacing of the colorbar'

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**CombinedBase**(*data=None, ax=None, auto_update=None, project=None, draw=None, make_plot=True, clear=False, enable_post=False, **kwargs*)
Bases: `psy_simple.plotters.ScalarCombinedBase`

Base plotter for combined 2-dimensional scalar and vector plot

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the `post` formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Vector plot formatoptions

`arrowsize`	Change the size of the arrows
`arrowstyle`	Change the style of the arrows

Methods

`check_data`(name, dims, is_unstructured)	A validation method for the data shape

Color coding formatoptions

`color`	Set the color for the arrows
`vbounds`	Specify the boundaries of the vector colorbar
`vcbar`	Specify the position of the vector plot colorbars
`vcbarspacing`	Specify the spacing of the bounds in the colorbar
`vcmap`	Specify the color map
`vctickprops`	Specify the font properties of the colorbar ticklabels
`vcticksize`	Specify the font size of the colorbar ticklabels
`vctickweight`	Specify the fontweight of the colorbar ticklabels
`bounds`	Specify the boundaries of the colorbar
`cbar`	Specify the position of the colorbars

Miscallaneous formatoptions

`linewidth`	Change the linewidth of the arrows

Label formatoptions

`vclabel`	Show the colorbar label of the vector plot
`vclabelprops`	Properties of the Vector colorbar label
`vclabelsize`	Set the size of the Vector colorbar label
`vclabelweight`	Set the fontweight of the Vector colorbar label

Axis tick formatoptions

`vcticklabels`	Specify the colorbar ticklabels
`vcticks`	Specify the tick locations of the vector colorbar
`cticks`	Specify the tick locations of the colorbar

Masking formatoptions

`maskbetween`	Mask data points between two numbers
`maskgeq`	Mask data points greater than or equal to a number
`maskgreater`	Mask data points greater than a number
`maskleq`	Mask data points smaller than or equal to a number
`maskless`	Mask data points smaller than a number

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

arrowsize
Change the size of the arrows

Possible types

- *None* – make no scaling
- *float* – Factor scaling the size of the arrows

See also:

arrowstyle, *linewidth*, `density`, *color*

arrowstyle
Change the style of the arrows

Possible types

str – Any arrow style string (see `FancyArrowPatch`)

Notes

This formatoption only has an effect for stream plots

See also:

arrowsize, *linewidth*, `density`, *color*

classmethod check_data (*name*, *dims*, *is_unstructured*)
A validation method for the data shape

Parameters

- **name** (*list of str with length 2*) – The variable names (one for the first, two for the second array)
- **dims** (*list with length 2 of lists with length 1*) – The dimension of the arrays. Only 2D-Arrays are allowed (or 1-D if an array is unstructured)
- **is_unstructured** (*bool or list of bool*) – True if the corresponding array is unstructured.

Returns

- *list of bool or None* – True, if everything is okay, False in case of a serious error, None if it is intermediate. Each object in this list corresponds to one in the given *name*
- *list of str* – The message giving more information on the reason. Each object in this list corresponds to one in the given *name*

color
Set the color for the arrows

This formatoption can be used to set a single color for the vectors or define the color coding

Possible types

- *float* – Determines the greyness
- *color* – Defines the same color for all arrows. The string can be either a html hex string (e.g. '#eeefff'), a single letter (e.g. 'b': blue, 'g': green, 'r': red, 'c': cyan, 'm': magenta, 'y': yellow, 'k': black, 'w': white) or any other color
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *2D-array* – The values determine the color for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

arrowsize, *arrowstyle*, `density`, *linewidth*

linewidth

Change the linewidth of the arrows

Possible types

- *float* – give the linewidth explicitly
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *tuple (string, float)* – *string* may be one of the above strings, *float* may be a scaling factor
- *2D-array* – The values determine the linewidth for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

arrowsize, *arrowstyle*, `density`, *color*

vbounds

Specify the boundaries of the vector colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

data plot the ticks exactly where the data is.

mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

`cmap` Specifies the colormap

`vcbar`

Specify the position of the vector plot colorbars

Possible types

- *bool* – True: defaults to ‘b’ False: Don’t draw any colorbar
- *str* – The string can be a combination of one of the following strings: {‘fr’, ‘fb’, ‘fl’, ‘ft’, ‘b’, ‘r’, ‘sv’, ‘sh’}
  - ‘b’, ‘r’ stand for bottom and right of the axes
  - ‘fr’, ‘fb’, ‘fl’, ‘ft’ stand for bottom, right, left and top of the figure
  - ‘sv’ and ‘sh’ stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

vcbarspacing
Specify the spacing of the bounds in the colorbar

Possible types

str {'uniform', 'proportional'} – if `'uniform'`, every color has exactly the same width in the colorbar, if `'proportional'`, the size is chosen according to the data

vclabel
Show the colorbar label of the vector plot

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `set_label()` method.

See also:

`vclabelsize`, *`vclabelweight`*, *`vclabelprops`*

vclabelprops
Properties of the Vector colorbar label

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`vclabel`, *`vclabelsize`*, *`vclabelweight`*

vclabelsize
Set the size of the Vector colorbar label

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`vclabel`, `vclabelweight`, `vclabelprops`

vclabelweight
Set the fontweight of the Vector colorbar label

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`vclabel`, `vclabelsize`, `vclabelprops`

vcmap
Specify the color map

This formatoption specifies the color coding of the data via a `matplotlib.colors.Colormap`

Possible types

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

`bounds` specifies the boundaries of the colormap

vcticklabels
Specify the colorbar ticklabels

Possible types

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`cticks`, `cticksize`, `ctickweight`, `ctickprops`, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

vctickprops
Specify the font properties of the colorbar ticklabels

Possible types

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`cticksize`, `ctickweight`, `cticklabels`, *`cticks`*, *`vcticksize`*, *`vctickweight`*, *`vcticklabels`*, *`vcticks`*

vcticks
Specify the tick locations of the vector colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  bounds let the *`bounds`* keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`, *`vcticklabels`*

vcticksize
Specify the font size of the colorbar ticklabels

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`ctickweight`, `ctickprops`, `cticklabels`, *`cticks`*, *`vctickweight`*, *`vctickprops`*, *`vcticklabels`*, *`vcticks`*

vctickweight
Specify the fontweight of the colorbar ticklabels

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`cticksize`, `ctickprops`, `cticklabels`, *`cticks`*, *`vcticksize`*, *`vctickprops`*, *`vcticklabels`*, *`vcticks`*

bounds
Specify the boundaries of the colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

`cmap` Specifies the colormap

`cbar`
Specify the position of the colorbars

Possible types

- *bool* – True: defaults to 'b' False: Don't draw any colorbar
- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

Examples

Draw a colorbar at the bottom and left of the axes:

```
>>> plotter.update(cbar='bl')
```

`maskbetween`
Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

`maskless`, *`maskleq`*, *`maskgreater`*, *`maskgeq`*

`maskgeq`
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgeq*, *maskbetween*

maskleq
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

post
Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post`

attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding Plotter to True

---

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the post formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

cticks

Specify the tick locations of the colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually

- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  bounds let the `bounds` keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`

class `psy_simple.plotters.`**`CombinedSimplePlotter`**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: `psy_simple.plotters.CombinedBase`, `psy_simple.plotters.Simple2DPlotter`, `psy_simple.plotters.SimpleVectorPlotter`

Combined 2D plotter and vector plotter

See also:

`psyplot.plotter.maps.CombinedPlotter` for visualizing the data on a map

Vector plot formatoptions

`density`	Change the density of the arrows
`arrowsize`	Change the size of the arrows
`arrowstyle`	Change the style of the arrows

Plot formatoptions

`plot`	Choose how to visualize a 2-dimensional scalar data field
`vplot`	Choose the vector plot type

Color coding formatoptions

bounds	Specify the boundaries of the colorbar
cbar	Specify the position of the colorbars
cbarspacing	Specify the spacing of the bounds in the colorbar
cmap	Specify the color map
color	Set the color for the arrows
ctickprops	Specify the font properties of the colorbar ticklabels
cticksize	Specify the font size of the colorbar ticklabels
ctickweight	Specify the fontweight of the colorbar ticklabels
extend	Draw arrows at the side of the colorbar
levels	The levels for the contour plot
miss_color	Set the color for missing values
vbounds	Specify the boundaries of the vector colorbar
vcbar	Specify the position of the vector plot colorbars
vcbarspacing	Specify the spacing of the bounds in the colorbar
vcmap	Specify the color map
vctickprops	Specify the font properties of the colorbar ticklabels
vcticksize	Specify the font size of the colorbar ticklabels
vctickweight	Specify the fontweight of the colorbar ticklabels

Masking formatoptions

maskbetween	Mask data points between two numbers
maskgeq	Mask data points greater than or equal to a number
maskgreater	Mask data points greater than a number
maskleq	Mask data points smaller than or equal to a number
maskless	Mask data points smaller than a number

Label formatoptions

clabel	Show the colorbar label
clabelprops	Properties of the Colorbar label
clabelsize	Set the size of the Colorbar label
clabelweight	Set the fontweight of the Colorbar label
figtitle	Plot a figure title
figtitleprops	Properties of the figure title
figtitlesize	Set the size of the figure title
figtitleweight	Set the fontweight of the figure title
labelprops	Set the font properties of both, x- and y-label
labelsize	Set the size of both, x- and y-label
labelweight	Set the font size of both, x- and y-label
text	Add text anywhere on the plot
title	Show the title
titleprops	Properties of the title
titlesize	Set the size of the title
titleweight	Set the fontweight of the title
vclabel	Show the colorbar label of the vector plot
vclabelprops	Properties of the Vector colorbar label
vclabelsize	Set the size of the Vector colorbar label
vclabelweight	Set the fontweight of the Vector colorbar label
xlabel	Set the x-axis label

Continued on next page

Table 117 – continued from previous page

ylabel	Set the y-axis label

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

Axes formatoptions

axiscolor	Color the x- and y-axes
grid	Display the grid
tight	Automatically adjust the plots.
transpose	Switch x- and y-axes
xlim	Set the x-axis limits
ylim	Set the y-axis limits

Axis tick formatoptions

cticklabels	Specify the colorbar ticklabels
cticks	Specify the tick locations of the colorbar
vcticklabels	Specify the colorbar ticklabels
vcticks	Specify the tick locations of the vector colorbar
xrotation	Rotate the x-axis ticks
xticklabels	Modify the x-axis ticklabels
xtickprops	Specify the x-axis tick parameters
xticks	Modify the x-axis ticks
yrotation	Rotate the y-axis ticks
yticklabels	Modify the y-axis ticklabels
ytickprops	Specify the y-axis tick parameters
yticks	Modify the y-axis ticks

Miscallaneous formatoptions

datagrid	Show the grid of the data
interp_bounds	Interpolate grid cell boundaries for 2D plots
linewidth	Change the linewidth of the arrows
sym_lims	Make x- and y-axis symmetric
ticksize	Change the ticksize of the ticklabels
tickweight	Change the fontweight of the ticks

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also

the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.

- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the `post` formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

density
Change the density of the arrows

Possible types

- *float* – Scales the density of the arrows in x- and y-direction (1.0 means no scaling)
- *tuple (x, y)* – Defines the scaling in x- and y-direction manually

Notes

quiver plots do not support density scaling

plot
Choose how to visualize a 2-dimensional scalar data field

Possible types

- *None* – Don't make any plotting
- *'mesh'* – Use the `matplotlib.pyplot.pcolormesh()` function to make the plot or the `matplotlib.pyplot.tripcolor()` for an unstructered grid
- *'tri'* – Use the `matplotlib.pyplot.tripcolor()` function to plot data on a triangular grid
- *'contourf'* – Make a filled contour plot using the `matplotlib.pyplot.contourf()` function or the `matplotlib.pyplot.tricontourf()` for triangular data. The levels for the contour plot are controlled by the `levels` formatoption
- *'tricontourf'* – Make a filled contour plot using the `matplotlib.pyplot.tricontourf()` function

vplot
Choose the vector plot type

Possible types

str – Plot types can be either

quiver to make a quiver plot

stream to make a stream plot

bounds
Specify the boundaries of the colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

`cmap` Specifies the colormap

cbar

Specify the position of the colorbars

Possible types

- *bool* – True: defaults to 'b' False: Don't draw any colorbar
- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

Examples

Draw a colorbar at the bottom and left of the axes:

```
>>> plotter.update(cbar='bl')
```

cbarspacing

Specify the spacing of the bounds in the colorbar

Possible types

str {'uniform', 'proportional'} – if `'uniform'`, every color has exactly the same width in the colorbar, if `'proportional'`, the size is chosen according to the data

cmap

Specify the color map

This formatoption specifies the color coding of the data via a `matplotlib.colors.Colormap`

Possible types

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

`bounds` specifies the boundaries of the colormap

color

Set the color for the arrows

This formatoption can be used to set a single color for the vectors or define the color coding

Possible types

- *float* – Determines the greyness
- *color* – Defines the same color for all arrows. The string can be either a html hex string (e.g. '#eeefff'), a single letter (e.g. 'b': blue, 'g': green, 'r': red, 'c': cyan, 'm': magenta, 'y': yellow, 'k': black, 'w': white) or any other color
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *2D-array* – The values determine the color for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

arrowsize, *arrowstyle*, *density*, *linewidth*

ctickprops

Specify the font properties of the colorbar ticklabels

Possible types

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

cticksize, *ctickweight*, *cticklabels*, *cticks*, *vcticksize*, *vctickweight*, *vcticklabels*, *vcticks*

cticksize

Specify the font size of the colorbar ticklabels

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

ctickweight, *ctickprops*, *cticklabels*, *cticks*, *vctickweight*, *vctickprops*, *vcticklabels*, *vcticks*

ctickweight

Specify the fontweight of the colorbar ticklabels

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

ctitcksize, *ctickprops*, *cticklabels*, *cticks*, *vcticksize*, *vctickprops*, *vcticklabels*, *vcticks*

extend
Draw arrows at the side of the colorbar

Possible types

str {'neither', 'both', 'min' or 'max'} – If not 'neither', make pointed end(s) for out-of-range values

levels
The levels for the contour plot

This formatoption sets the levels for the filled contour plot and only has an effect if the *plot* Formatoption is set to `'contourf'`

Possible types

- *None* – Use the settings from the *bounds* formatoption and if this does not specify boundaries, use 11
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.

miss_color
Set the color for missing values

Possible types

- *None* – Use the default from the colormap
- *string, tuple.* – Defines the color of the grid.

vbounds
Specify the boundaries of the vector colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

cmap Specifies the colormap

vcbar
Specify the position of the vector plot colorbars

Possible types

- *bool* – True: defaults to 'b' False: Don't draw any colorbar
- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

vcbarspacing
Specify the spacing of the bounds in the colorbar

Possible types

str {'uniform', 'proportional'} – if `'uniform'`, every color has exactly the same width in the colorbar, if `'proportional'`, the size is chosen according to the data

vcmap
Specify the color map

This formatoption specifies the color coding of the data via a `matplotlib.colors.Colormap`

Possible types

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

bounds specifies the boundaries of the colormap

vctickprops
Specify the font properties of the colorbar ticklabels

Possible types

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

cticksize, *ctickweight*, *cticklabels*, *cticks*, *vcticksize*, *vctickweight*, *vcticklabels*, *vcticks*

vcticksize
Specify the font size of the colorbar ticklabels

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

ctickweight, *ctickprops*, *cticklabels*, *cticks*, *vctickweight*, *vctickprops*, *vcticklabels*, *vcticks*

vctickweight
Specify the fontweight of the colorbar ticklabels

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

cticksize, *ctickprops*, *cticklabels*, *cticks*, *vcticksize*, *vctickprops*, *vcticklabels*, *vcticks*

maskbetween
Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskgeq*

maskgeq
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgeq*, *maskbetween*

maskleq
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

clabel
Show the colorbar label

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `set_label()` method.

See also:

clabelsize, *clabelweight*, *clabelprops*

clabelprops
Properties of the Colorbar label

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

clabel, *clabelsize*, *clabelweight*

clabelsize
Set the size of the Colorbar label

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

clabel, *clabelweight*, *clabelprops*

clabelweight
Set the fontweight of the Colorbar label

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

clabel, *clabelsize*, *clabelprops*

figtitle
Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)

- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this `Figtitle` instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *`title`* formatoption.

See also:

`title`, *`figtitlesize`*, *`figtitleweight`*, *`figtitleprops`*

`figtitleprops`
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`figtitle`, *`figtitlesize`*, *`figtitleweight`*

`figtitlesize`
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`figtitle`, *`figtitleweight`*, *`figtitleprops`*

figtitleweight
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

figtitle, *figtitlesize*, *figtitleprops*

labelprops
Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

xlabel, *ylabel*, *labelsize*, *labelweight*

labelsize
Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

xlabel, *ylabel*, *labelweight*, *labelprops*

labelweight
Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

xlabel, *ylabel*, *labelsize*, *labelprops*

text

Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

title, *figtitle*

title

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the *`figtitle`* formatoption.

See also:

`figtitle`, *`titlesize`*, *`titleweight`*, *`titleprops`*

`titleprops`
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`title`, *`titlesize`*, *`titleweight`*

`titlesize`
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`title`, *`titleweight`*, *`titleprops`*

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

title, *titlesize*, *titleprops*

vclabel
Show the colorbar label of the vector plot

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `set_label()` method.

See also:

vclabelsize, *vclabelweight*, *vclabelprops*

vclabelprops
Properties of the Vector colorbar label

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

vclabel, *vclabelsize*, *vclabelweight*

vclabelsize

Set the size of the Vector colorbar label

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

vclabel, *vclabelweight*, *vclabelprops*

vclabelweight

Set the fontweight of the Vector colorbar label

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

vclabel, *vclabelsize*, *vclabelprops*

xlabel

Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

ylabel

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post`

attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

axiscolor

Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

grid
Display the grid

Show the grid on the plot with the specified color.

Possible types

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

tight
Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

> **Warning:** There is no update method to undo what happend after this formatoption is set to True!

`transpose`
Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

`xlim`
Set the x-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

`ylim`
Set the y-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`xlim`

`cticklabels`
Specify the colorbar ticklabels

Possible types

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`cticks`, `cticksize`, `ctickweight`, `ctickprops`, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

`cticks`
Specify the tick locations of the colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

bounds let the `bounds` keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`

vcticklabels
Specify the colorbar ticklabels

Possible types

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`cticks`, `cticksize`, `ctickweight`, `ctickprops`, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

vcticks
Specify the tick locations of the vector colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, …]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

bounds let the `bounds` keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`, `vcticklabels`

xrotation
Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`yrotation`

xticklabels
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

xtickprops
Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.

- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *ytickprops*

xticks
Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

`xticklabels`, `ticksize`, `tickweight`, `xtickprops`, `yticks`

yrotation
Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`xrotation`

yticklabels
Modify the y-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`yticks`, `ticksize`, `tickweight`, `ytickprops`, `xticklabels`

ytickprops
Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.

- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *xtickprops*

yticks

Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

yticklabels, *ticksize*, *tickweight*, *ytickprops* *xticks* for possible examples

arrowsize
Change the size of the arrows

Possible types

- *None* – make no scaling
- *float* – Factor scaling the size of the arrows

See also:

arrowstyle, *linewidth*, *density*, *color*

arrowstyle
Change the style of the arrows

Possible types

str – Any arrow style string (see `FancyArrowPatch`)

Notes

This formatoption only has an effect for stream plots

See also:

arrowsize, *linewidth*, *density*, *color*

datagrid
Show the grid of the data

This formatoption shows the grid of the data (without labels)

Possible types

- *None* – Don't show the data grid
- *str* – A linestyle in the form `'k-'`, where `'k'` is the color and `'-'` the linestyle.
- *dict* – any keyword arguments that are passed to the plotting function (`matplotlib.pyplot.triplot()` for triangular grids and `matplotlib.pyplot.hlines()` for rectilinear grids)

interp_bounds
Interpolate grid cell boundaries for 2D plots

This formatoption can be used to tell enable and disable the interpolation of grid cell boundaries. Usually, netCDF files only contain the centered coordinates. In this case, we interpolate the boundaries between the grid cell centers.

Possible types

- *None* – Interpolate the boundaries, except for circumpolar grids
- *bool* – If True (the default), the grid cell boundaries are inter- and extrapolated. Otherwise, if False, the coordinate centers are used and the default behaviour of matplotlib cuts of the most outer row and column of the 2D-data. Note that this results in a slight shift of the data

linewidth
Change the linewidth of the arrows

Possible types

- *float* – give the linewidth explicitly
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *tuple (string, float)* – *string* may be one of the above strings, *float* may be a scaling factor
- *2D-array* – The values determine the linewidth for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

arrowsize, *arrowstyle*, *density*, *color*

sym_lims
Make x- and y-axis symmetric

Possible types

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

ticksize
Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)

- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`tickweight`, `xtickprops`, `ytickprops`

tickweight
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`ticksize`, `xtickprops`, `ytickprops`

class psy_simple.plotters.**CombinedVectorPlot**(**args*, ***kwargs*)
Bases: `psy_simple.plotters.VectorPlot`

Choose the vector plot type

Possible types

Attributes

`arrowsize`	arrowsize Formatoption instance in the plotter
`arrowstyle`	arrowstyle Formatoption instance in the plotter
`bounds`	bounds Formatoption instance in the plotter
`cmap`	cmap Formatoption instance in the plotter
`color`	color Formatoption instance in the plotter
`data_dependent`	bool(x) -> bool
`density`	density Formatoption instance in the plotter
`linewidth`	linewidth Formatoption instance in the plotter
`transform`	transform Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter

Methods

`update`(*args, **kwargs)	Method that is call to update the formatoption on the axes

str – Plot types can be either

quiver to make a quiver plot

stream to make a stream plot

arrowsize
arrowsize Formatoption instance in the plotter

arrowstyle
arrowstyle Formatoption instance in the plotter

bounds
bounds Formatoption instance in the plotter

cmap
cmap Formatoption instance in the plotter

color
color Formatoption instance in the plotter

data_dependent = True

density
density Formatoption instance in the plotter

linewidth
linewidth Formatoption instance in the plotter

transform
transform Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

update(**args*, ***kwargs*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**ContourLevels**(**args*, ***kwargs*)
Bases: `psy_simple.plotters.Bounds`

The levels for the contour plot

This formatoption sets the levels for the filled contour plot and only has an effect if the `plot` Formatoption is set to `'contourf'`

Possible types

Attributes

`cbar`	cbar Formatoption instance in the plotter
`cbounds`	cbounds Formatoption instance in the plotter
`cmap`	cmap Formatoption instance in the plotter
`dependencies`	list() -> new empty list
`name`	str(object='') -> str
`priority`	int(x=0) -> integer

Methods

`update`(value)	Method that is call to update the formatoption on the axes

- *None* – Use the settings from the `bounds` formatoption and if this does not specify boundaries, use 11
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.

cbar
cbar Formatoption instance in the plotter

cbounds
cbounds Formatoption instance in the plotter

cmap
cmap Formatoption instance in the plotter

dependencies = ['cbounds']

name = 'Levels for the filled contour plot'

priority = 20

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**DataGrid**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Show the grid of the data

This formatoption shows the grid of the data (without labels)

Possible types

Attributes

`array`	The (masked) data array that is plotted
`children`	list() -> new empty list

Continued on next page

Table 126 – continued from previous page

name	str(object='') -> str
transform	transform Formatoption instance in the plotter
triangles	The `matplotlib.tri.Triangulation` instance containing the
xbounds	Boundaries of the x-coordinate
ybounds	Boundaries of the y-coordinate

Methods

remove()	Method to remove the effects of this formatoption
update(value)	Method that is call to update the formatoption on the axes

- *None* – Don't show the data grid
- *str* – A linestyle in the form `'k-'`, where `'k'` is the color and `'-'` the linestyle.
- *dict* – any keyword arguments that are passed to the plotting function (`matplotlib.pyplot.triplot()` for triangular grids and `matplotlib.pyplot.hlines()` for rectilinear grids)

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the *children* attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the *children*, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

array
The (masked) data array that is plotted

children = ['transform']

name = 'Grid of the data'

remove()
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

transform
transform Formatoption instance in the plotter

triangles
The `matplotlib.tri.Triangulation` instance containing the spatial informations

update (*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

xbounds
Boundaries of the x-coordinate

ybounds
Boundaries of the y-coordinate

class psy_simple.plotters.**DataPrecision** (*key, plotter=None, index_in_list=None, additional_children=[], additional_dependencies=[], **kwargs*)
Bases: `psyplot.plotter.Formatoption`

Set the precision of the data

This formatoption can be used to specify the precision of the data which then will be the minimal bin width of the 2D histogram or the bandwith of the kernel size (if the `density` formatoption is set to `'kde'`)

Possible types

Attributes

`connections`	list() -> new empty list
`data_dependent`	bool(x) -> bool
`density`	density Formatoption instance in the plotter
`dependencies`	list() -> new empty list
`group`	str(object='') -> str
`name`	str(object='') -> str
`priority`	int(x=0) -> integer
`xrange`	xrange Formatoption instance in the plotter
`yrange`	yrange Formatoption instance in the plotter

Methods

`update`(value)	Method that is call to update the formatoption on the axes

- *float* – If 0, this formatoption has no effect at all. Otherwise it is assumed to be the precision of the data
- *str* – One of `{'scott' | 'silverman'}`. If the `density` formatoption is set to `'kde'`, this describes the method how to calculate the bandwidth

Parameters

- **key** (`str`) – formatoption key in the *plotter*
- **plotter** (`psyplot.plotter.Plotter`) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (`int or None`) – The index that shall be used if the data is a `psyplot.InteractiveList`

- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, *dependencies* and *connections* attributes, with values being the name of the new formatoption in this plotter.

connections = ['density']

data_dependent = True

density
density Formatoption instance in the plotter

dependencies = ['xrange', 'yrange']

group = 'data'

name = 'Precision of the visualized data'

priority = 30

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

xrange
xrange Formatoption instance in the plotter

yrange
yrange Formatoption instance in the plotter

class psy_simple.plotters.**DataTicksCalculator**(**args*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Abstract base formatoption to calculate ticks and bounds from the data

Possible types

Attributes

array	The numpy array of the data
data_dependent	bool(x) -> bool
full_array	The full array of this and the shared data

- *numeric array* – specifies the ticks manually
- *str or list [str, …]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

array
The numpy array of the data

data_dependent = True

full_array
The full array of this and the shared data

class psy_simple.plotters.**Density**(**args*, ***kwargs*)
Bases: psyplot.plotter.Formatoption

Change the density of the arrows

Possible types

Attributes

data_dependent	bool(x) -> bool
dependencies	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
priority	int(x=0) -> integer

Methods

remove([plot_type])	Method to remove the effects of this formatoption
update(value)	Method that is call to update the formatoption on the axes

- *float* – Scales the density of the arrows in x- and y-direction (1.0 means no scaling)
- *tuple (x, y)* – Defines the scaling in x- and y-direction manually

Notes

quiver plots do not support density scaling

data_dependent = True

dependencies = ['plot']

group = 'vector'

name = 'Density of the arrows'

plot
plot Formatoption instance in the plotter

priority = 20

remove (*plot_type=None*)
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

update (*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**DensityPlotter** (*data=None, ax=None, auto_update=None, project=None, draw=None, make_plot=True, clear=False, enable_post=False, **kwargs*)

Bases: *psy_simple.plotters.Simple2DPlotter*

A plotter to visualize the density of points in a 2-dimensional grid

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the *post* formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Attributes

allowed_dims	int(x=0) -> integer
allowed_vars	int(x=0) -> integer

Data manipulation formatoptions

bins	Specify the bins of the 2D-Histogramm
coord	Use an alternative variable as x-coordinate

Continued on next page

Table 134 – continued from previous page

density	Specify the method to calculate the density
normed	Specify the normalization of the histogram
precision	Set the precision of the data
xrange	Specify the range of the histogram for the x-dimension
yrange	Specify the range of the histogram for the x-dimension

Color coding formatoptions

bounds	Specify the boundaries of the colorbar
cbar	Specify the position of the colorbars
cbarspacing	Specify the spacing of the bounds in the colorbar
cmap	Specify the color map
ctickprops	Specify the font properties of the colorbar ticklabels
cticksize	Specify the font size of the colorbar ticklabels
ctickweight	Specify the fontweight of the colorbar ticklabels
extend	Draw arrows at the side of the colorbar
levels	The levels for the contour plot
miss_color	Set the color for missing values

Masking formatoptions

maskbetween	Mask data points between two numbers
maskgeq	Mask data points greater than or equal to a number
maskgreater	Mask data points greater than a number
maskleq	Mask data points smaller than or equal to a number
maskless	Mask data points smaller than a number

Label formatoptions

clabel	Show the colorbar label
clabelprops	Properties of the Colorbar label
clabelsize	Set the size of the Colorbar label
clabelweight	Set the fontweight of the Colorbar label
figtitle	Plot a figure title
figtitleprops	Properties of the figure title
figtitlesize	Set the size of the figure title
figtitleweight	Set the fontweight of the figure title
labelprops	Set the font properties of both, x- and y-label
labelsize	Set the size of both, x- and y-label
labelweight	Set the font size of both, x- and y-label
text	Add text anywhere on the plot
title	Show the title
titleprops	Properties of the title
titlesize	Set the size of the title
titleweight	Set the fontweight of the title
xlabel	Set the x-axis label
ylabel	Set the y-axis label

Post processing formatoptions

`post`	Apply your own postprocessing script
`post_timing`	Determine when to run the `post` formatoption

Axes formatoptions

`axiscolor`	Color the x- and y-axes
`grid`	Display the grid
`tight`	Automatically adjust the plots.
`transpose`	Switch x- and y-axes
`xlim`	Set the x-axis limits
`ylim`	Set the y-axis limits

Axis tick formatoptions

`cticklabels`	Specify the colorbar ticklabels
`cticks`	Specify the tick locations of the colorbar
`xrotation`	Rotate the x-axis ticks
`xticklabels`	Modify the x-axis ticklabels
`xtickprops`	Specify the x-axis tick parameters
`xticks`	Modify the x-axis ticks
`yrotation`	Rotate the y-axis ticks
`yticklabels`	Modify the y-axis ticklabels
`ytickprops`	Specify the y-axis tick parameters
`yticks`	Modify the y-axis ticks

Miscallaneous formatoptions

`datagrid`	Show the grid of the data
`interp_bounds`	Interpolate grid cell boundaries for 2D plots
`sym_lims`	Make x- and y-axis symmetric
`ticksize`	Change the ticksize of the ticklabels
`tickweight`	Change the fontweight of the ticks

Plot formatoptions

`plot`	Specify the plotting method

allowed_dims = 1

allowed_vars = 1

bins

Specify the bins of the 2D-Histogramm

This formatoption can be used to specify, how many bins to use. In other words, it determines the grid size of the resulting histogram or kde plot. If however you also set the `precision` formatoption keyword then the minimum of precision and the bins specified here will be used.

Possible types

- *int* – If 0, only use the bins specified by the `precision` keyword (raises an error if the `precision` is also set to 0), otherwise the number of bins to use
- *tuple (x, y) of int* – The bins for x and y explicitly

coord

Use an alternative variable as x-coordinate

This formatoption let's you specify another variable in the base dataset of the data array in case you want to use this as the x-coordinate instead of the raw data

Possible types

- *None* – Use the default
- *str* – The name of the variable to use in the base dataset
- *xarray.DataArray* – An alternative variable with the same shape as the displayed array

---

Examples

To see the difference, we create a simple test dataset:

```
>>> import xarray as xr

>>> import numpy as np

>>> import psyplot.project as psy

>>> ds = xr.Dataset({
...     'temp': xr.Variable(('time', ), np.arange(5)),
...     'std': xr.Variable(('time', ), np.arange(5, 10))})
>>> ds
<xarray.Dataset>
Dimensions:  (time: 5)
Coordinates:
  * time     (time) int64 0 1 2 3 4
Data variables:
    temp     (time) int64 0 1 2 3 4
    std      (time) int64 5 6 7 8 9
```

If we create a plot with it, we get the `'time'` dimension on the x-axis:

```
>>> plotter = psy.plot.lineplot(ds, name=['temp']).plotters[0]

>>> plotter.plot_data[0].dims
('time',)
```

If we however set the `'coord'` keyword, we get:

```
>>> plotter = psy.plot.lineplot(
...     ds, name=['temp'], coord='std').plotters[0]

>>> plotter.plot_data[0].dims
('std',)
```

and `'std'` is plotted on the x-axis.

---

density

Specify the method to calculate the density

Possible types

str – One of the following strings are possible

hist Make a 2D-histogram. The normalization is controlled by the *normed* formatoption

kde Fit a bivariate kernel density estimate to the data. Note that this choice requires pythons *[statsmodels]* module to be installed

References

normed

Specify the normalization of the histogram

This formatoption can be used to normalize the histogram. It has no effect if the *density* formatoption is set to `'kde'`

Possible types

- *None* – Do not make any normalization
- *str* – One of

  counts To make the normalization based on the total number counts

  area To make the normalization basen on the total number of counts and area (the default behaviour of `numpy.histogram2d()`)

See also:

density

precision

Set the precision of the data

This formatoption can be used to specify the precision of the data which then will be the minimal bin width of the 2D histogram or the bandwith of the kernel size (if the *density* formatoption is set to `'kde'`)

Possible types

- *float* – If 0, this formatoption has no effect at all. Otherwise it is assumed to be the precision of the data
- *str* – One of `{'scott' | 'silverman'}`. If the *density* formatoption is set to `'kde'`, this describes the method how to calculate the bandwidth

xrange

Specify the range of the histogram for the x-dimension

This formatoption specifies the minimum and maximum of the histogram in the x-dimension

Possible types

- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

Notes

This formatoption always acts on the coordinate, no matter what the value of the `transpose` formatoption is

See also:

`yrange`

yrange

Specify the range of the histogram for the x-dimension

This formatoption specifies the minimum and maximum of the histogram in the x-dimension

Possible types

- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

Notes

This formatoption always acts on the DataArray, no matter what the value of the `transpose` formatoption is

See also:

`xrange`

bounds
Specify the boundaries of the colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

cmap Specifies the colormap

cbar

Specify the position of the colorbars

Possible types

- *bool* – True: defaults to 'b' False: Don't draw any colorbar
- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

Examples

Draw a colorbar at the bottom and left of the axes:

```
>>> plotter.update(cbar='bl')
```

cbarspacing

Specify the spacing of the bounds in the colorbar

Possible types

str {'uniform', 'proportional'} – if `'uniform'`, every color has exactly the same width in the colorbar, if `'proportional'`, the size is chosen according to the data

cmap

Specify the color map

This formatoption specifies the color coding of the data via a `matplotlib.colors.Colormap`

Possible types

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

bounds specifies the boundaries of the colormap

ctickprops
Specify the font properties of the colorbar ticklabels

Possible types

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

ctSize, *ctickweight*, *cticklabels*, *cticks*, vcticksize, vctickweight, vcticklabels, vcticks

cticksize
Specify the font size of the colorbar ticklabels

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

ctickweight, *ctickprops*, *cticklabels*, *cticks*, vctickweight, vctickprops, vcticklabels, vcticks

ctickweight
Specify the fontweight of the colorbar ticklabels

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

cticksize, *ctickprops*, *cticklabels*, *cticks*, vcticksize, vctickprops, vcticklabels, vcticks

extend
Draw arrows at the side of the colorbar

Possible types

str {'neither', 'both', 'min' or 'max'} – If not 'neither', make pointed end(s) for out-of-range values

levels
The levels for the contour plot

This formatoption sets the levels for the filled contour plot and only has an effect if the *plot* Formatoption is set to `'contourf'`

Possible types

- *None* – Use the settings from the `bounds` formatoption and if this does not specify boundaries, use 11
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.

`miss_color`
Set the color for missing values

Possible types

- *None* – Use the default from the colormap
- *string, tuple.* – Defines the color of the grid.

`maskbetween`
Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

`maskless`, `maskleq`, `maskgreater`, `maskgeq`

`maskgeq`
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgeq*, *maskbetween*

maskleq
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

clabel
Show the colorbar label

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`

– sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `set_label()` method.

See also:

clabelsize, *clabelweight*, *clabelprops*

clabelprops
Properties of the Colorbar label

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

clabel, *clabelsize*, *clabelweight*

clabelsize
Set the size of the Colorbar label

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

clabel, *clabelweight*, *clabelprops*

clabelweight
Set the fontweight of the Colorbar label

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

clabel, *clabelsize*, *clabelprops*

figtitle
Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this `Figtitle` instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *`title`* formatoption.

See also:

`title`, *`figtitlesize`*, *`figtitleweight`*, *`figtitleprops`*

`figtitleprops`
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`figtitle`, *`figtitlesize`*, *`figtitleweight`*

`figtitlesize`
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

figtitle, *figtitleweight*, *figtitleprops*

figtitleweight
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

figtitle, *figtitlesize*, *figtitleprops*

labelprops
Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

xlabel, *ylabel*, *labelsize*, *labelweight*

labelsize
Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

xlabel, *ylabel*, *labelweight*, *labelprops*

labelweight
Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

xlabel, *ylabel*, *labelsize*, *labelprops*

text
Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

title, *figtitle*

title

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the *figtitle* formatoption.

See also:

figtitle, *titlesize*, *titleweight*, *titleprops*

titleprops

Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

title, *titlesize*, *titleweight*

titlesize

Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

title, *titleweight*, *titleprops*

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

title, *titlesize*, *titleprops*

xlabel
Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

ylabel

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

axiscolor

Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

grid

Display the grid

Show the grid on the plot with the specified color.

Possible types

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

tight

Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

> **Warning:** There is no update method to undo what happend after this formatoption is set to True!

transpose
Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

xlim
Set the x-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

ylim
Set the y-axis limits

Possible types

- *None* – To not change the current limits

- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`xlim`

`cticklabels`

Specify the colorbar ticklabels

Possible types

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`cticks`, `cticksize`, `ctickweight`, `ctickprops`, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

`cticks`

Specify the tick locations of the colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

bounds let the `bounds` keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`

xrotation
Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`yrotation`

xticklabels
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

xtickprops
Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.

- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *ytickprops*

xticks
Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

xticklabels, *ticksize*, *tickweight*, *xtickprops*, *yticks*

yrotation
Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

xrotation

yticklabels
Modify the y-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

yticks, *ticksize*, *tickweight*, *ytickprops*, *xticklabels*

ytickprops
Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.

- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *xtickprops*

yticks

Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

yticklabels, *ticksize*, *tickweight*, *ytickprops*

xticks for possible examples

datagrid

Show the grid of the data

This formatoption shows the grid of the data (without labels)

Possible types

- *None* – Don't show the data grid
- *str* – A linestyle in the form `'k-'`, where `'k'` is the color and `'-'` the linestyle.
- *dict* – any keyword arguments that are passed to the plotting function (`matplotlib.pyplot.triplot()` for triangular grids and `matplotlib.pyplot.hlines()` for rectilinear grids)

interp_bounds

Interpolate grid cell boundaries for 2D plots

This formatoption can be used to tell enable and disable the interpolation of grid cell boundaries. Usually, netCDF files only contain the centered coordinates. In this case, we interpolate the boundaries between the grid cell centers.

Possible types

- *None* – Interpolate the boundaries, except for circumpolar grids
- *bool* – If True (the default), the grid cell boundaries are inter- and extrapolated. Otherwise, if False, the coordinate centers are used and the default behaviour of matplotlib cuts of the most outer row and column of the 2D-data. Note that this results in a slight shift of the data

sym_lims

Make x- and y-axis symmetric

Possible types

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

ticksize

Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)

- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

tickweight, *xtickprops*, *ytickprops*

tickweight
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

ticksize, *xtickprops*, *ytickprops*

plot
Specify the plotting method

Possible types

- *None* – Don't make any plotting
- *'mesh'* – Use the `matplotlib.pyplot.pcolormesh()` function to make the plot

class psy_simple.plotters.**DtTicksBase**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.TicksBase*, *psy_simple.plotters.TicksManager*

Abstract base class for x- and y-tick formatoptions

Possible types

Attributes

dtdata	The np.unique `data` as datetime objects
plot	plot Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter

Methods

update(value)	Method that is call to update the formatoption on the axes

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.

- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

dtdata
The np.unique `data` as datetime objects

plot
plot Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**ErrorAlpha**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Set the alpha value for the error range

This formatoption can be used to set the alpha value (opacity) for the *error* formatoption

Possible types

Attributes

`connections`	list() -> new empty list
`error`	error Formatoption instance in the plotter
`group`	str(object='') -> str
`name`	str(object='') -> str
`priority`	int(x=0) -> integer

Methods

`update`(value)	Method that is call to update the formatoption on the axes

float – A float between 0 and 1

See also:

`error`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

connections = ['error']

error
error Formatoption instance in the plotter

group = 'colors'

name = 'Alpha value of the error range'

priority = 20

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**ErrorCalculator**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: psyplot.plotter.Formatoption

Calculation of the error

This formatoption is used to calculate the error range.

Possible types

Attributes

children	list() -> new empty list
data_dependent	bool(x) -> bool
group	str(object='') -> str
mean	mean Formatoption instance in the plotter
name	str(object='') -> str
priority	int(x=0) -> integer
requires_replot	bool(x) -> bool

Methods

update(value)	Method that is call to update the formatoption on the axes

- *None* – Do not calculate any error range
- *float* – A float between 0 and 50. This will represent the distance from the median (i.e. the 50th percentile). A value of 45 will hence correspond to the 5th and 95th percentile
- *list of 2 floats between 0 and 100* – Two floats where the first corresponds to the minimum and the second to the maximum percentile
- *str* – A string with 'std' in it. Then we will use the standard deviation. Any number in this string, e.g. '3.5std' will serve as a multiplier (in this case 3.5 times the standard deviation).

See also:

mean Determines how the line is calculated

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the *children* attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the dependencies attribute)

- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the *children*, dependencies and connections attributes, with values being the name of the new formatoption in this plotter.

children = ['mean']

data_dependent = True

group = 'data'

mean
mean Formatoption instance in the plotter

name = 'Mean calculation'

priority = 30

requires_replot = True

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**ErrorPlot**(**args*, ***kwargs*)
Bases: psyplot.plotter.Formatoption

Visualize the error range

This formatoption visualizes the error range. For this, you must provide a two-dimensional data array as input. The first dimension might be either of length

- 2 to provide the deviation from minimum and maximum error range from the data
- 3 to provide the minimum and maximum error range explicitly

Attributes

children	list() -> new empty list
color	color Formatoption instance in the plotter
data_dependent	bool(x) -> bool
group	str(object='') -> str
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
plot_fmt	bool(x) -> bool
priority	int(x=0) -> integer
transpose	transpose Formatoption instance in the plotter

Methods

make_plot()	
plot_fill(index, min_range, max_range, c, …)	
remove()	Method to remove the effects of this formatoption
update(value)	Method that is call to update the formatoption on the axes

Possible types

- *None* – No errors are visualized
- *'fill'* – The area between min- and max-error is filled with the same color as the line and the alpha is determined by the `fillalpha` attribute

---

Examples

Assume you have the standard deviation stored in the `'std'`-variable and the data in the `'data'` variable. Then you can visualize the standard deviation simply via:

```
>>> psy.plot.lineplot(input_ds, name=[['data', 'std']])
```

On the other hand, assume you want to visualize the area between the 25th and 75th percentile (stored in the variables `'p25'` and `'p75'`):

```
>>> psy.plot.lineplot(input_ds, name=[['data', 'p25', 'p75']])
```

See also:

`erroralpha`

`children = ['color', 'transpose', 'plot']`

`color`
color Formatoption instance in the plotter

`data_dependent = True`

`group = 'plotting'`

`make_plot`()

`name = 'Error plot type'`

`plot`
plot Formatoption instance in the plotter

`plot_fill`(*index*, *min_range*, *max_range*, *c*, ***kwargs*)

`plot_fmt = True`

`priority = 20`

`remove`()
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

`transpose`
transpose Formatoption instance in the plotter

`update`(*value*)
Method that is call to update the formatoption on the axes

Parameters **`value`** – Value to update

class psy_simple.plotters.**Extend**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Draw arrows at the side of the colorbar

Possible types

Attributes

connections	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
plot	plot Formatoption instance in the plotter

Methods

update(value)	Method that is call to update the formatoption on the axes

str {'neither', 'both', 'min' or 'max'} – If not 'neither', make pointed end(s) for out-of-range values

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and *connections* attributes, with values being the name of the new formatoption in this plotter.

connections = ['plot']

group = 'colors'

name = 'Ends of the colorbar'

plot

plot Formatoption instance in the plotter

update(*value*)

Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**FldmeanPlotter**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: `psy_simple.plotters.LinePlotter`

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the `post` formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Attributes

`allowed_dims`	int(x=0) -> integer

Data manipulation formatoptions

`coord`	Use an alternative variable as x-coordinate
`err_calc`	Calculation of the error
`mean`	Determine how the error is visualized

Masking formatoptions

`maskbetween`	Mask data points between two numbers
`maskgeq`	Mask data points greater than or equal to a number
`maskgreater`	Mask data points greater than a number
`maskleq`	Mask data points smaller than or equal to a number
`maskless`	Mask data points smaller than a number

Color coding formatoptions

`color`	Set the color coding
`erroralpha`	Set the alpha value for the error range

Label formatoptions

`figtitle`	Plot a figure title
`figtitleprops`	Properties of the figure title
`figtitlesize`	Set the size of the figure title
`figtitleweight`	Set the fontweight of the figure title
`labelprops`	Set the font properties of both, x- and y-label
`labelsize`	Set the size of both, x- and y-label
`labelweight`	Set the font size of both, x- and y-label
`text`	Add text anywhere on the plot
`title`	Show the title
`titleprops`	Properties of the title
`titlesize`	Set the size of the title
`titleweight`	Set the fontweight of the title
`xlabel`	Set the x-axis label
`ylabel`	Set the y-axis label

Post processing formatoptions

`post`	Apply your own postprocessing script
`post_timing`	Determine when to run the `post` formatoption

Axes formatoptions

`axiscolor`	Color the x- and y-axes
`grid`	Display the grid
`tight`	Automatically adjust the plots.
`transpose`	Switch x- and y-axes
`xlim`	Set the x-axis limits
`ylim`	Set the y-axis limits

Axis tick formatoptions

`xrotation`	Rotate the x-axis ticks
`xticklabels`	Modify the x-axis ticklabels
`xtickprops`	Specify the x-axis tick parameters
`xticks`	Modify the x-axis ticks
`yrotation`	Rotate the y-axis ticks
`yticklabels`	Modify the y-axis ticklabels
`ytickprops`	Specify the y-axis tick parameters
`yticks`	Modify the y-axis ticks

Miscallaneous formatoptions

`legend`	Draw a legend
`legendlabels`	Set the labels of the arrays in the legend
`linewidth`	Choose the width of the lines
`marker`	Choose the marker for points
`markersize`	Choose the size of the markers for points
`sym_lims`	Make x- and y-axis symmetric
`ticksize`	Change the ticksize of the ticklabels

Continued on next page

Table 161 – continued from previous page

tickweight	Change the fontweight of the ticks

Plot formatoptions

error	Visualize the error range
plot	Choose the line style of the plot

allowed_dims = 3

coord

Use an alternative variable as x-coordinate

This formatoption let's you specify another variable in the base dataset of the data array in case you want to use this as the x-coordinate instead of the raw data

Possible types

- *None* – Use the default
- *str* – The name of the variable to use in the base dataset
- *xarray.DataArray* – An alternative variable with the same shape as the displayed array

---

Examples

To see the difference, we create a simple test dataset:

```
>>> import xarray as xr

>>> import numpy as np

>>> import psyplot.project as psy

>>> ds = xr.Dataset({
...     'temp': xr.Variable(('time', ), np.arange(5)),
...     'std': xr.Variable(('time', ), np.arange(5, 10))})
>>> ds
<xarray.Dataset>
Dimensions:  (time: 5)
Coordinates:
  * time     (time) int64 0 1 2 3 4
Data variables:
    temp     (time) int64 0 1 2 3 4
    std      (time) int64 5 6 7 8 9
```

If we create a plot with it, we get the `'time'` dimension on the x-axis:

```
>>> plotter = psy.plot.lineplot(ds, name=['temp']).plotters[0]

>>> plotter.plot_data[0].dims
('time',)
```

If we however set the `'coord'` keyword, we get:

```
>>> plotter = psy.plot.lineplot(
...     ds, name=['temp'], coord='std').plotters[0]

>>> plotter.plot_data[0].dims
('std',)
```

and `'std'` is plotted on the x-axis.

---

err_calc
Calculation of the error

This formatoption is used to calculate the error range.

Possible types

- *None* – Do not calculate any error range
- *float* – A float between 0 and 50. This will represent the distance from the median (i.e. the 50th percentile). A value of 45 will hence correspond to the 5th and 95th percentile
- *list of 2 floats between 0 and 100* – Two floats where the first corresponds to the minimum and the second to the maximum percentile
- *str* – A string with 'std' in it. Then we will use the standard deviation. Any number in this string, e.g. '3.5std' will serve as a multiplier (in this case 3.5 times the standard deviation).

See also:

mean Determines how the line is calculated

maskbetween
Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskgeq*

maskgeq
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

`maskless`, `maskleq`, `maskgeq`, `maskbetween`

`maskleq`
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

`maskless`, `maskgreater`, `maskgeq`, `maskbetween`

`maskless`
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

`maskleq`, `maskgreater`, `maskgeq`, `maskbetween`

`mean`
Determine how the error is visualized

Possible types

- *'mean'* – Calculate the weighted mean
- *'median'* – Calculate the weighted median (i.e. the 50th percentile)
- *float between 0 and 100* – Calculate the given quantile

See also:

`err_calc` Determines how to calculate the error

`color`
Set the color coding

This formatoptions sets the color of the lines, bars, etc.

Possible types

- *None* – to use the axes color_cycle
- *iterable* – (e.g. list) to specify the colors manually

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.ColorMap* – to automatically choose the colors according to the number of lines, etc. from the given colormap

erroralpha

Set the alpha value for the error range

This formatoption can be used to set the alpha value (opacity) for the *error* formatoption

Possible types

float – A float between 0 and 1

See also:

error

figtitle

Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this `Figtitle` instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.

- This is the title of the whole figure! For the title of this specific subplot, see the *title* formatoption.

See also:

title, *figtitlesize*, *figtitleweight*, *figtitleprops*

figtitleprops
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

figtitle, *figtitlesize*, *figtitleweight*

figtitlesize
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

figtitle, *figtitleweight*, *figtitleprops*

figtitleweight
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

figtitle, *figtitlesize*, *figtitleprops*

labelprops
Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

xlabel, *ylabel*, *labelsize*, *labelweight*

labelsize
Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

xlabel, *ylabel*, *labelweight*, *labelprops*

labelweight
Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

xlabel, *ylabel*, *labelsize*, *labelprops*

text
Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`

- desc: `%(long_name)s [%(units)s]`
- dinfo: `%B %d, %Y`
- tinfo: `%H:%M`
- sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

title, *figtitle*

title

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the *figtitle* formatoption.

See also:

figtitle, *titlesize*, *titleweight*, *titleprops*

titleprops
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

title, *titlesize*, *titleweight*

titlesize
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

title, *titleweight*, *titleprops*

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

title, *titlesize*, *titleprops*

xlabel
Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)

- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

ylabel

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts

- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

`post` The post processing formatoption

`axiscolor`

Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

`grid`

Display the grid

Show the grid on the plot with the specified color.

Possible types

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

tight
Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

> **Warning:** There is no update method to undo what happend after this formatoption is set to True!

transpose
Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

xlim
Set the x-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

minmax Uses the minimum and maximum

sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

`ylim`
Set the y-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`xlim`

`xrotation`
Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`yrotation`

`xticklabels`
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

xticks, *ticksize*, *tickweight*, *xtickprops*, *yticklabels*

xtickprops
Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *ytickprops*

xticks
Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

hour draw ticks every hour

day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

xticklabels, *ticksize*, *tickweight*, *xtickprops*, *yticks*

yrotation
Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

xrotation

yticklabels
Modify the y-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

yticks, *ticksize*, *tickweight*, *ytickprops*, *xticklabels*

ytickprops
Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *xtickprops*

yticks
Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

hour draw ticks every hour

day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

`yticklabels`, `ticksize`, `tickweight`, `ytickprops`

`xticks` for possible examples

`legend`

Draw a legend

This formatoption determines where and if to draw the legend. It uses the `labels` formatoption to determine the labels.

Possible types

- *bool* – Draw a legend or not
- *str or int* – Specifies where to plot the legend (i.e. the location)
- *dict* – Give the keywords for the `matplotlib.pyplot.legend()` function

See also:

`labels`

`legendlabels`

Set the labels of the arrays in the legend

This formatoption specifies the labels for each array in the legend. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)

- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – A single string that shall be used for all arrays.
- *list of str* – Same as a single string but specified for each array

See also:

legend

linewidth

Choose the width of the lines

Possible types

- *None* – Use the default from matplotlibs rcParams
- *float* – The width of the lines

marker

Choose the marker for points

Possible types

- *None* – Use the default from matplotlibs rcParams
- *str* – A valid symbol for the matplotlib markers (see `matplotlib.markers`)

markersize

Choose the size of the markers for points

Possible types

- *None* – Use the default from matplotlibs rcParams
- *float* – The size of the marker

sym_lims

Make x- and y-axis symmetric

Possible types

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

`ticksize`
Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`tickweight`, *`xtickprops`*, *`ytickprops`*

`tickweight`
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`ticksize`, *`xtickprops`*, *`ytickprops`*

`error`
Visualize the error range

This formatoption visualizes the error range. For this, you must provide a two-dimensional data array as input. The first dimension might be either of length

- 2 to provide the deviation from minimum and maximum error range from the data
- 3 to provide the minimum and maximum error range explicitly

Possible types

- *None* – No errors are visualized
- *'fill'* – The area between min- and max-error is filled with the same color as the line and the alpha is determined by the `fillalpha` attribute

---

Examples

Assume you have the standard deviation stored in the `'std'`-variable and the data in the `'data'` variable. Then you can visualize the standard deviation simply via:

```
>>> psy.plot.lineplot(input_ds, name=[['data', 'std']])
```

On the other hand, assume you want to visualize the area between the 25th and 75th percentile (stored in the variables `'p25'` and `'p75'`):

```
>>> psy.plot.lineplot(input_ds, name=[['data', 'p25', 'p75']])
```

See also:

`erroralpha`

`plot`
Choose the line style of the plot

Possible types

- *None* – Don't make any plotting
- `'area'` – To make an area plot (filled between y=0 and y), see `matplotlib.pyplot.fill_between()`
- `'areax'` – To make a transposed area plot (filled between x=0 and x), see `matplotlib.pyplot.fill_betweenx()`
- *str or list of str* – The line style string to use (['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '–' | '-.' | ':' | 'None' | ' ' | '']).

class `psy_simple.plotters.`**`Grid`**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Display the grid

Show the grid on the plot with the specified color.

Possible types

Attributes

`group`	str(object='') -> str
`name`	str(object='') -> str

Methods

`update`(value)	Method that is call to update the formatoption on the axes

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

group = 'axes'

name = 'Grid lines'

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**Hist2DXRange**(**args*, ***kwargs*)

Bases: *psy_simple.plotters.LimitBase*

Specify the range of the histogram for the x-dimension

This formatoption specifies the minimum and maximum of the histogram in the x-dimension

Possible types

Attributes

array	The numpy array of the data
coord	coord Formatoption instance in the plotter
data_dependent	bool(x) -> bool
dependencies	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
priority	int(x=0) -> integer
transpose	transpose Formatoption instance in the plotter

Methods

set_limit(*args)	The method to set the minimum and maximum limit

- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

Notes

This formatoption always acts on the coordinate, no matter what the value of the *transpose* formatoption is

See also:

`yrange`

array
The numpy array of the data

coord
coord Formatoption instance in the plotter

data_dependent = True

dependencies = ['coord']

group = 'data'

name = 'Range of the histogram in x-direction'

plot
plot Formatoption instance in the plotter

priority = 30

set_limit(**args*)
The method to set the minimum and maximum limit

Parameters

- **min_val** (*float*) – The value for the lower limit
- **max_val** (*float*) – The value for the upper limit

transpose
transpose Formatoption instance in the plotter

class psy_simple.plotters.**Hist2DYRange**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.Hist2DXRange*

Specify the range of the histogram for the x-dimension

This formatoption specifies the minimum and maximum of the histogram in the x-dimension

Possible types

Attributes

array	The numpy array of the data
coord	coord Formatoption instance in the plotter
data_dependent	bool(x) -> bool
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter

- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

Notes

This formatoption always acts on the DataArray, no matter what the value of the *transpose* formatoption is

See also:

`xrange`

array
The numpy array of the data

coord
coord Formatoption instance in the plotter

data_dependent = True

name = 'Range of the histogram in y-direction'

plot
plot Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

class psy_simple.plotters.**HistBins**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Specify the bins of the 2D-Histogramm

This formatoption can be used to specify, how many bins to use. In other words, it determines the grid size of the resulting histogram or kde plot. If however you also set the *precision* formatoption keyword then the minimum of precision and the bins specified here will be used.

Possible types

Attributes

data_dependent	bool(x) -> bool
dependencies	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
precision	precision Formatoption instance in the plotter
priority	int(x=0) -> integer

Methods

update(value)	Method that is call to update the formatoption on the axes

- *int* – If 0, only use the bins specified by the *precision* keyword (raises an error if the *precision* is also set to 0), otherwise the number of bins to use
- *tuple (x, y) of int* – The bins for x and y explicitly

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, *dependencies* and `connections` attributes, with values being the name of the new formatoption in this plotter.

data_dependent = True

dependencies = ['precision']

group = 'data'

name = 'Number of bins of the histogram'

precision
precision Formatoption instance in the plotter

priority = 30

update (*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**InterpolateBounds** (*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Interpolate grid cell boundaries for 2D plots

This formatoption can be used to tell enable and disable the interpolation of grid cell boundaries. Usually, netCDF files only contain the centered coordinates. In this case, we interpolate the boundaries between the grid cell centers.

Possible types

Methods

update(value)	Method that is call to update the formatoption on the axes

- *None* – Interpolate the boundaries, except for circumpolar grids
- *bool* – If True (the default), the grid cell boundaries are inter- and extrapolated. Otherwise, if False, the coordinate centers are used and the default behaviour of matplotlib cuts of the most outer row and column of the 2D-data. Note that this results in a slight shift of the data

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

update(*value*)

Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**LabelOptions**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.DictFormatoption`

Base formatoption class for label sizes

Possible types

Attributes

children	list() -> new empty list
xlabel	xlabel Formatoption instance in the plotter
ylabel	ylabel Formatoption instance in the plotter

Methods

update(value)	Method that is call to update the formatoption on the axes
update_axis(value)	

dict – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.

- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

children = ['xlabel', 'ylabel']

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

update_axis(*value*)

xlabel
xlabel Formatoption instance in the plotter

ylabel
ylabel Formatoption instance in the plotter

class psy_simple.plotters.**LabelProps**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: `psy_simple.plotters.LabelOptions`

Set the font properties of both, x- and y-label

Possible types

Attributes

`children`	list() -> new empty list
`group`	str(object='') -> str
`labelsize`	labelsize Formatoption instance in the plotter
`labelweight`	labelweight Formatoption instance in the plotter
`name`	str(object='') -> str
`xlabel`	xlabel Formatoption instance in the plotter
`ylabel`	ylabel Formatoption instance in the plotter

Methods

`update_axis`(fontprops)

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

xlabel, *ylabel*, *labelsize*, *labelweight*

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the *children* attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the *children*, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

children = ['xlabel', 'ylabel', 'labelsize', 'labelweight']

group = 'labels'

labelsize
labelsize Formatoption instance in the plotter

labelweight
labelweight Formatoption instance in the plotter

name = 'font properties of x- and y-axis label'

update_axis (*fontprops*)

xlabel
xlabel Formatoption instance in the plotter

ylabel
ylabel Formatoption instance in the plotter

class psy_simple.plotters.**LabelSize**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.LabelOptions*

Set the size of both, x- and y-label

Possible types

Attributes

group	str(object=’‘) -> str
labelprops	labelprops Formatoption instance in the plotter
name	str(object=’‘) -> str
parents	list() -> new empty list

Continued on next page

Table 175 – continued from previous page

xlabel	xlabel Formatoption instance in the plotter
ylabel	ylabel Formatoption instance in the plotter

Methods

update_axis(value)

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

xlabel, *ylabel*, `labelweight`, *labelprops*

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

group = 'labels'

labelprops
labelprops Formatoption instance in the plotter

name = 'font size of x- and y-axis label'

parents = ['labelprops']

update_axis(*value*)

xlabel
xlabel Formatoption instance in the plotter

ylabel
ylabel Formatoption instance in the plotter

class psy_simple.plotters.**LabelWeight**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.LabelOptions*

Set the font size of both, x- and y-label

Possible types

Attributes

group	str(object='') -> str
labelprops	labelprops Formatoption instance in the plotter
name	str(object='') -> str
parents	list() -> new empty list
xlabel	xlabel Formatoption instance in the plotter
ylabel	ylabel Formatoption instance in the plotter

Methods

update_axis(value)

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

xlabel, *ylabel*, `labelsize`, *labelprops*

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

group = 'labels'

labelprops
labelprops Formatoption instance in the plotter

name = 'font weight of x- and y-axis label'

parents = ['labelprops']

update_axis(*value*)

xlabel
xlabel Formatoption instance in the plotter

ylabel
ylabel Formatoption instance in the plotter

class psy_simple.plotters.**Legend**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: `psyplot.plotter.DictFormatoption`

Draw a legend

This formatoption determines where and if to draw the legend. It uses the `labels` formatoption to determine the labels.

Possible types

Attributes

`color`	color Formatoption instance in the plotter
`dependencies`	list() -> new empty list
`legendlabels`	legendlabels Formatoption instance in the plotter
`marker`	marker Formatoption instance in the plotter
`name`	str(object='') -> str
`plot`	plot Formatoption instance in the plotter

Methods

`get_artists_and_labels`()	
`remove`()	Method to remove the effects of this formatoption
`update`(value)	Method that is call to update the formatoption on the axes

- *bool* – Draw a legend or not
- *str or int* – Specifies where to plot the legend (i.e. the location)
- *dict* – Give the keywords for the `matplotlib.pyplot.legend()` function

See also:

`labels`

Parameters

- **key** (`str`) – formatoption key in the *plotter*
- **plotter** (`psyplot.plotter.Plotter`) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.

- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

color
color Formatoption instance in the plotter

dependencies = ['legendlabels', 'plot', 'color', 'marker']

get_artists_and_labels()

legendlabels
legendlabels Formatoption instance in the plotter

marker
marker Formatoption instance in the plotter

name = 'Properties of the legend'

plot
plot Formatoption instance in the plotter

remove()
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**LegendLabels**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`, `psy_simple.base.TextBase`

Set the labels of the arrays in the legend

This formatoption specifies the labels for each array in the legend. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).

- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Attributes

data_dependent	bool(x) -> bool
name	str(object='') -> str

Methods

update(value)	Method that is call to update the formatoption on the axes

Possible types

- *str* – A single string that shall be used for all arrays.
- *list of str* – Same as a single string but specified for each array

See also:

`legend`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

data_dependent = True

name = 'Labels in the legend'

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **`value`** – Value to update

class `psy_simple.plotters.`**`LimitBase`**(**args*, ***kwargs*)

Bases: `psy_simple.plotters.DataTicksCalculator`

Base class for x- and y-limits

Possible types

Attributes

`children`	list() -> new empty list
`connections`	list() -> new empty list
`group`	str(object='') -> str
`plot`	plot Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter
`value2share`	The value that is passed to shared formatoptions (by default, the

Methods

`set_limit`(min_val, max_val)	The method to set the minimum and maximum limit
`update`(value)	Method that is call to update the formatoption on the axes

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

`children = ['transpose']`

`connections = ['plot']`

`group = 'axes'`

`plot`

plot Formatoption instance in the plotter

`set_limit`(*min_val*, *max_val*)

The method to set the minimum and maximum limit

Parameters

- **min_val** (*float*) – The value for the lower limit
- **max_val** (*float*) – The value for the upper limit

transpose
transpose Formatoption instance in the plotter

update (*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

value2share
The value that is passed to shared formatoptions (by default, the `value` attribute)

class psy_simple.plotters.**LineColors** (**args*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Set the color coding

This formatoptions sets the color of the lines, bars, etc.

Possible types

Attributes

group	str(object='') -> str
name	str(object='') -> str
priority	int(x=0) -> integer
value2pickle	The value that can be used when pickling the information of the project
value2share	The value that is passed to shared formatoptions (by default, the

Methods

update(value)	Method that is call to update the formatoption on the axes

- *None* – to use the axes color_cycle
- *iterable* – (e.g. list) to specify the colors manually
- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.ColorMap* – to automatically choose the colors according to the number of lines, etc. from the given colormap

group = 'colors'

name = 'Color cycle'

priority = 20

update (*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

value2pickle
The value that can be used when pickling the information of the project

value2share
The value that is passed to shared formatoptions (by default, the `value` attribute)

class psy_simple.plotters.**LinePlot**(**args*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Choose the line style of the plot

Possible types

Attributes

children	list() -> new empty list
color	color Formatoption instance in the plotter
data_dependent	bool(x) -> bool
group	str(object='') -> str
marker	marker Formatoption instance in the plotter
name	str(object='') -> str
plot_fmt	bool(x) -> bool
priority	float(x) -> floating point number
transpose	transpose Formatoption instance in the plotter

Methods

make_plot()	
plot_arr(arr, c, ls, m)	
remove()	Method to remove the effects of this formatoption
update(value)	Method that is call to update the formatoption on the axes

- *None* – Don't make any plotting
- `'area'` – To make an area plot (filled between y=0 and y), see `matplotlib.pyplot.fill_between()`
- `'areax'` – To make a transposed area plot (filled between x=0 and x), see `matplotlib.pyplot.fill_betweenx()`
- *str or list of str* – The line style string to use (['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '–' | '-.' | ':' | 'None' | ' ' | '']).

children = ['color', 'transpose', 'marker']

color
color Formatoption instance in the plotter

data_dependent = True

group = 'plotting'

make_plot()

marker
marker Formatoption instance in the plotter

name = 'Line plot type'

plot_arr (*arr*, *c*, *ls*, *m*)

plot_fmt = True

priority = 20.1

remove ()
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

transpose
transpose Formatoption instance in the plotter

update (*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**LinePlotter** (*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)
Bases: *psy_simple.plotters.SimplePlotterBase*

Plotter for simple one-dimensional line plots

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the *post* formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Attributes

allowed_vars	The number variables that one data array visualized by this plotter might have.

Data manipulation formatoptions

coord	Use an alternative variable as x-coordinate

Plot formatoptions

error	Visualize the error range
plot	Choose the line style of the plot

Color coding formatoptions

erroralpha	Set the alpha value for the error range
color	Set the color coding

Miscallaneous formatoptions

linewidth	Choose the width of the lines
marker	Choose the marker for points
markersize	Choose the size of the markers for points
legend	Draw a legend
legendlabels	Set the labels of the arrays in the legend
sym_lims	Make x- and y-axis symmetric
ticksize	Change the ticksize of the ticklabels
tickweight	Change the fontweight of the ticks

Masking formatoptions

maskbetween	Mask data points between two numbers
maskgeq	Mask data points greater than or equal to a number
maskgreater	Mask data points greater than a number
maskleq	Mask data points smaller than or equal to a number
maskless	Mask data points smaller than a number

Label formatoptions

figtitle	Plot a figure title
figtitleprops	Properties of the figure title
figtitlesize	Set the size of the figure title
figtitleweight	Set the fontweight of the figure title
labelprops	Set the font properties of both, x- and y-label
labelsize	Set the size of both, x- and y-label
labelweight	Set the font size of both, x- and y-label
text	Add text anywhere on the plot
title	Show the title
titleprops	Properties of the title

Continued on next page

Table 195 – continued from previous page

titlesize	Set the size of the title
titleweight	Set the fontweight of the title
xlabel	Set the x-axis label
ylabel	Set the y-axis label

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

Axes formatoptions

axiscolor	Color the x- and y-axes
grid	Display the grid
tight	Automatically adjust the plots.
transpose	Switch x- and y-axes
xlim	Set the x-axis limits
ylim	Set the y-axis limits

Axis tick formatoptions

xrotation	Rotate the x-axis ticks
xticklabels	Modify the x-axis ticklabels
xtickprops	Specify the x-axis tick parameters
xticks	Modify the x-axis ticks
yrotation	Rotate the y-axis ticks
yticklabels	Modify the y-axis ticklabels
ytickprops	Specify the y-axis tick parameters
yticks	Modify the y-axis ticks

allowed_vars = 3

The number variables that one data array visualized by this plotter might have. We allow up to 3 variableswhere the second and third variable might be the errors (see the *error* formatoption)

coord

Use an alternative variable as x-coordinate

This formatoption let's you specify another variable in the base dataset of the data array in case you want to use this as the x-coordinate instead of the raw data

Possible types

- *None* – Use the default
- *str* – The name of the variable to use in the base dataset
- *xarray.DataArray* – An alternative variable with the same shape as the displayed array

Examples

To see the difference, we create a simple test dataset:

```
>>> import xarray as xr

>>> import numpy as np

>>> import psyplot.project as psy

>>> ds = xr.Dataset({
...     'temp': xr.Variable(('time', ), np.arange(5)),
...     'std': xr.Variable(('time', ), np.arange(5, 10))})
>>> ds
<xarray.Dataset>
Dimensions:  (time: 5)
Coordinates:
  * time     (time) int64 0 1 2 3 4
Data variables:
    temp     (time) int64 0 1 2 3 4
    std      (time) int64 5 6 7 8 9
```

If we create a plot with it, we get the `'time'` dimension on the x-axis:

```
>>> plotter = psy.plot.lineplot(ds, name=['temp']).plotters[0]

>>> plotter.plot_data[0].dims
('time',)
```

If we however set the `'coord'` keyword, we get:

```
>>> plotter = psy.plot.lineplot(
...     ds, name=['temp'], coord='std').plotters[0]

>>> plotter.plot_data[0].dims
('std',)
```

and `'std'` is plotted on the x-axis.

error

Visualize the error range

This formatoption visualizes the error range. For this, you must provide a two-dimensional data array as input. The first dimension might be either of length

- 2 to provide the deviation from minimum and maximum error range from the data
- 3 to provide the minimum and maximum error range explicitly

Possible types

- *None* – No errors are visualized
- *'fill'* – The area between min- and max-error is filled with the same color as the line and the alpha is determined by the `fillalpha` attribute

Examples

Assume you have the standard deviation stored in the `'std'`-variable and the data in the `'data'` variable. Then you can visualize the standard deviation simply via:

```
>>> psy.plot.lineplot(input_ds, name=[['data', 'std']])
```

On the other hand, assume you want to visualize the area between the 25th and 75th percentile (stored in the variables `'p25'` and `'p75'`):

```
>>> psy.plot.lineplot(input_ds, name=[['data', 'p25', 'p75']])
```

See also:

erroralpha

erroralpha

Set the alpha value for the error range

This formatoption can be used to set the alpha value (opacity) for the *error* formatoption

Possible types

float – A float between 0 and 1

See also:

error

linewidth

Choose the width of the lines

Possible types

- *None* – Use the default from matplotlibs rcParams
- *float* – The width of the lines

marker

Choose the marker for points

Possible types

- *None* – Use the default from matplotlibs rcParams
- *str* – A valid symbol for the matplotlib markers (see `matplotlib.markers`)

markersize

Choose the size of the markers for points

Possible types

- *None* – Use the default from matplotlibs rcParams
- *float* – The size of the marker

plot

Choose the line style of the plot

Possible types

- *None* – Don't make any plotting
- `'area'` – To make an area plot (filled between y=0 and y), see `matplotlib.pyplot.fill_between()`
- `'areax'` – To make a transposed area plot (filled between x=0 and x), see `matplotlib.pyplot.fill_betweenx()`
- *str or list of str* – The line style string to use (['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | '-' | '–' | '-.' | ':' | 'None' | ' ' | '']).

color
Set the color coding

This formatoptions sets the color of the lines, bars, etc.

Possible types

- *None* – to use the axes color_cycle
- *iterable* – (e.g. list) to specify the colors manually
- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.ColorMap* – to automatically choose the colors according to the number of lines, etc. from the given colormap

maskbetween
Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskgeq*

maskgeq
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgeq*, *maskbetween*

maskleq
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

figtitle
Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this `Figtitle` instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *`title`* formatoption.

See also:

`title`, *`figtitlesize`*, *`figtitleweight`*, *`figtitleprops`*

`figtitleprops`
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`figtitle`, *`figtitlesize`*, *`figtitleweight`*

`figtitlesize`
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`figtitle`, *`figtitleweight`*, *`figtitleprops`*

`figtitleweight`
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`figtitle`, *`figtitlesize`*, *`figtitleprops`*

labelprops

Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

`xlabel`, *`ylabel`*, *`labelsize`*, *`labelweight`*

labelsize

Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`xlabel`, *`ylabel`*, *`labelweight`*, *`labelprops`*

labelweight

Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`xlabel`, *`ylabel`*, *`labelsize`*, *`labelprops`*

text

Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

`title`, *`figtitle`*

`title`

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`

- tinfo: `%H:%M`
- sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the `figtitle` formatoption.

See also:

`figtitle`, `titlesize`, `titleweight`, `titleprops`

titleprops
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`title`, `titlesize`, `titleweight`

titlesize
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`title`, `titleweight`, `titleprops`

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`title`, `titlesize`, `titleprops`

xlabel

Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

ylabel

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *`post_timing`* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

`post_timing` Determine the timing of this formatoption

post_timing

Determine when to run the *`post`* formatoption

This formatoption determines, whether the *`post`* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

`post` The post processing formatoption

axiscolor

Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

grid

Display the grid

Show the grid on the plot with the specified color.

Possible types

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

tight
Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

Warning: There is no update method to undo what happend after this formatoption is set to True!

transpose
Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

xlim
Set the x-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

minmax Uses the minimum and maximum

sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

`ylim`
Set the y-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

minmax Uses the minimum and maximum

sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`xlim`

`xrotation`
Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`yrotation`

xticklabels
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

xticks, *ticksize*, *tickweight*, *xtickprops*, *yticklabels*

xtickprops
Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *ytickprops*

xticks
Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

data plot the ticks exactly where the data is.

mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

hour draw ticks every hour

day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

`xticklabels`, `ticksize`, `tickweight`, `xtickprops`, `yticks`

yrotation
Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`xrotation`

yticklabels
Modify the y-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`yticks`, *`ticksize`*, *`tickweight`*, *`ytickprops`*, *`xticklabels`*

ytickprops
Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`xticks`, *`yticks`*, *`ticksize`*, *`tickweight`*, *`xtickprops`*

yticks
Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

data plot the ticks exactly where the data is.

mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

hour draw ticks every hour

day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

`yticklabels`, `ticksize`, `tickweight`, `ytickprops`

`xticks` for possible examples

`legend`

Draw a legend

This formatoption determines where and if to draw the legend. It uses the `labels` formatoption to determine the labels.

Possible types

- *bool* – Draw a legend or not
- *str or int* – Specifies where to plot the legend (i.e. the location)
- *dict* – Give the keywords for the `matplotlib.pyplot.legend()` function

See also:

`labels`

`legendlabels`

Set the labels of the arrays in the legend

This formatoption specifies the labels for each array in the legend. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.

- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – A single string that shall be used for all arrays.
- *list of str* – Same as a single string but specified for each array

See also:

legend

sym_lims

Make x- and y-axis symmetric

Possible types

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

ticksize

Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

tickweight, *xtickprops*, *ytickprops*

tickweight
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`ticksize`, `xtickprops`, `ytickprops`

class psy_simple.plotters.**LineWidth**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Choose the width of the lines

Possible types

Attributes

`connections`	list() -> new empty list
`plot`	plot Formatoption instance in the plotter
`priority`	int(x=0) -> integer

Methods

`update`(value)	Method that is call to update the formatoption on the axes

- *None* – Use the default from matplotlibs rcParams
- *float* – The width of the lines

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)

- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and *connections* attributes, with values being the name of the new formatoption in this plotter.

connections = ['plot']

plot
plot Formatoption instance in the plotter

priority = 20

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**Marker**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Choose the marker for points

Possible types

Attributes

priority	int(x=0) -> integer

Methods

update(value)	Method that is call to update the formatoption on the axes

- *None* – Use the default from matplotlibs rcParams
- *str* – A valid symbol for the matplotlib markers (see `matplotlib.markers`)

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords

may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

priority = 20

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**MarkerSize**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Choose the size of the markers for points

Possible types

Attributes

connections	list() -> new empty list
plot	plot Formatoption instance in the plotter
priority	int(x=0) -> integer

Methods

update(value)	Method that is call to update the formatoption on the axes

- *None* – Use the default from matplotlibs rcParams
- *float* – The size of the marker

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and *connections* attributes, with values being the name of the new formatoption in this plotter.

connections = ['plot']

plot
plot Formatoption instance in the plotter

priority = 20

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**MeanCalculator**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: psyplot.plotter.Formatoption

Determine how the error is visualized

Possible types

Attributes

data_dependent	bool(x) -> bool
group	str(object='') -> str
name	str(object='') -> str
priority	int(x=0) -> integer
requires_replot	bool(x) -> bool

Methods

update(value)	Method that is call to update the formatoption on the axes

- *'mean'* – Calculate the weighted mean
- *'median'* – Calculate the weighted median (i.e. the 50th percentile)
- *float between 0 and 100* – Calculate the given quantile

See also:

err_calc Determines how to calculate the error

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the dependencies attribute)

- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

data_dependent = True

group = 'data'

name = 'Mean calculation'

priority = 30

requires_replot = True

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**MissColor**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Set the color for missing values

Possible types

Attributes

`connections`	list() -> new empty list
`dependencies`	list() -> new empty list
`group`	str(object='') -> str
`name`	str(object='') -> str
`plot`	plot Formatoption instance in the plotter
`priority`	int(x=0) -> integer
`transform`	transform Formatoption instance in the plotter
`triangles`	The `matplotlib.tri.Triangulation` instance containing the
`update_after_plot`	bool(x) -> bool

Methods

`remove`()	Method to remove the effects of this formatoption
`update`(value)	Method that is call to update the formatoption on the axes

- *None* – Use the default from the colormap
- *string, tuple.* – Defines the color of the grid.

Parameters

- **key** (`str`) – formatoption key in the *plotter*
- **plotter** (`psyplot.plotter.Plotter`) – Plotter instance that holds this formatop-

tion. If None, it is assumed that this instance serves as a descriptor.

- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the children, *dependencies* and *connections* attributes, with values being the name of the new formatoption in this plotter.

connections = ['transform']

dependencies = ['plot']

group = 'colors'

name = 'Color of missing values'

plot
plot Formatoption instance in the plotter

priority = 10

remove()
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with requires_clearing set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual matplotlib.axes.Axes.clear() method.

transform
transform Formatoption instance in the plotter

triangles
The matplotlib.tri.Triangulation instance containing the spatial informations

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

update_after_plot = True

class psy_simple.plotters.**NormedHist2D**(*key, plotter=None, index_in_list=None, additional_children=[], additional_dependencies=[], **kwargs*)
Bases: psyplot.plotter.Formatoption

Specify the normalization of the histogram

This formatoption can be used to normalize the histogram. It has no effect if the density formatoption is set to 'kde'

Possible types

Attributes

data_dependent	bool(x) -> bool
group	str(object='') -> str
name	str(object='') -> str
priority	int(x=0) -> integer

Methods

hist2d(da, **kwargs)	Make the two dimensional histogram
update(value)	Method that is call to update the formatoption on the axes

- *None* – Do not make any normalization
- *str* – One of

  counts To make the normalization based on the total number counts

  area To make the normalization basen on the total number of counts and area (the default behaviour of `numpy.histogram2d()`)

See also:

`density`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

data_dependent = True

group = 'data'

hist2d(*da*, ***kwargs*)
Make the two dimensional histogram

Parameters da (*xarray.DataArray*) – The data source

name = 'Normalize the histogram'

priority = 30

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**Plot2D**(**args*, ***kwargs*)

Bases: psyplot.plotter.Formatoption

Choose how to visualize a 2-dimensional scalar data field

Possible types

Methods

add2format_coord(x, y)	Additional information for the format_coord()
get_xyz_1d(xcoord, x, ycoord, y, data)	Get closest x, y and z for the given *x* and *y* in *data* for
get_xyz_2d(xcoord, x, ycoord, y, data)	Get closest x, y and z for the given *x* and *y* in *data* for
get_xyz_tri(xcoord, x, ycoord, y, data)	Get closest x, y and z for the given *x* and *y* in *data* for
make_plot()	
remove()	Method to remove the effects of this formatoption
update(value)	Method that is call to update the formatoption on the axes

Attributes

array	The (masked) data array that is plotted
bounds	bounds Formatoption instance in the plotter
children	list() -> new empty list
cmap	cmap Formatoption instance in the plotter
data_dependent	bool(x) -> bool
dependencies	list() -> new empty list
format_coord	The function that can replace the axes.format_coord method
group	str(object=' ') -> str
interp_bounds	interp_bounds Formatoption instance in the plotter
levels	levels Formatoption instance in the plotter
mappable	Returns the mappable that can be used for colorbars
name	str(object=' ') -> str
plot_fmt	bool(x) -> bool
priority	int(x=0) -> integer
triangles	The matplotlib.tri.Triangulation instance containing the
xbounds	Boundaries of the x-coordinate
xcoord	The x coordinate xarray.Variable
ybounds	Boundaries of the y-coordinate
ycoord	The y coordinate xarray.Variable

- *None* – Don't make any plotting
- *'mesh'* – Use the matplotlib.pyplot.pcolormesh() function to make the plot or the matplotlib.pyplot.tripcolor() for an unstructered grid
- *'tri'* – Use the matplotlib.pyplot.tripcolor() function to plot data on a triangular grid
- *'contourf'* – Make a filled contour plot using the matplotlib.pyplot.contourf() function or the matplotlib.pyplot.tricontourf() for triangular data. The levels for the contour plot are

controlled by the `levels` formatoption

- *'tricontourf'* – Make a filled contour plot using the `matplotlib.pyplot.tricontourf()` function

add2format_coord (*x*, *y*)
Additional information for the `format_coord()`

array
The (masked) data array that is plotted

bounds
bounds Formatoption instance in the plotter

children = ['cmap', 'bounds']

cmap
cmap Formatoption instance in the plotter

data_dependent = True

dependencies = ['levels', 'interp_bounds']

format_coord
The function that can replace the axes.format_coord method

get_xyz_1d (*xcoord*, *x*, *ycoord*, *y*, *data*)
Get closest x, y and z for the given *x* and *y* in *data* for 1d coords

get_xyz_2d (*xcoord*, *x*, *ycoord*, *y*, *data*)
Get closest x, y and z for the given *x* and *y* in *data* for 2d coords

get_xyz_tri (*xcoord*, *x*, *ycoord*, *y*, *data*)
Get closest x, y and z for the given *x* and *y* in *data* for 1d coords

group = 'plotting'

interp_bounds
interp_bounds Formatoption instance in the plotter

levels
levels Formatoption instance in the plotter

make_plot ()

mappable
Returns the mappable that can be used for colorbars

name = '2D plot type'

plot_fmt = True

priority = 20

remove ()
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

triangles
The `matplotlib.tri.Triangulation` instance containing the spatial informations

update (*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

xbounds
Boundaries of the x-coordinate

xcoord
The x coordinate `xarray.Variable`

ybounds
Boundaries of the y-coordinate

ycoord
The y coordinate `xarray.Variable`

class psy_simple.plotters.**PointDensity**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Specify the method to calculate the density

Possible types

Attributes

bins	bins Formatoption instance in the plotter
coord	coord Formatoption instance in the plotter
data_dependent	bool(x) -> bool
dependencies	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
normed	normed Formatoption instance in the plotter
precision	precision Formatoption instance in the plotter
priority	int(x=0) -> integer
xrange	xrange Formatoption instance in the plotter
yrange	yrange Formatoption instance in the plotter

Methods

update(value)	Method that is call to update the formatoption on the axes

str – One of the following strings are possible

hist Make a 2D-histogram. The normalization is controlled by the *normed* formatoption

kde Fit a bivariate kernel density estimate to the data. Note that this choice requires pythons *[statsmodels]* module to be installed

References

Parameters

- **key** (*str*) – formatoption key in the *plotter*

- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, *dependencies* and `connections` attributes, with values being the name of the new formatoption in this plotter.

bins
bins Formatoption instance in the plotter

coord
coord Formatoption instance in the plotter

data_dependent = True

dependencies = ['normed', 'bins', 'xrange', 'yrange', 'precision', 'coord']

group = 'data'

name = 'Calculation of the point density'

normed
normed Formatoption instance in the plotter

precision
precision Formatoption instance in the plotter

priority = 30

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

xrange
xrange Formatoption instance in the plotter

yrange
yrange Formatoption instance in the plotter

class psy_simple.plotters.**ScalarCombinedBase**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: `psyplot.plotter.Plotter`

Base plotter for combined 2-dimensional scalar field with any other plotter

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself

- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the `post` formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Color coding formatoptions

`bounds`	Specify the boundaries of the colorbar
`cbar`	Specify the position of the colorbars

Axis tick formatoptions

`cticks`	Specify the tick locations of the colorbar

Masking formatoptions

`maskbetween`	Mask data points between two numbers
`maskgeq`	Mask data points greater than or equal to a number
`maskgreater`	Mask data points greater than a number
`maskleq`	Mask data points smaller than or equal to a number
`maskless`	Mask data points smaller than a number

Post processing formatoptions

`post`	Apply your own postprocessing script
`post_timing`	Determine when to run the `post` formatoption

bounds
Specify the boundaries of the colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines

the number of ticks (see below). A string can be one of the following:

data plot the ticks exactly where the data is.

mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

`cmap` Specifies the colormap

`cbar`
Specify the position of the colorbars

Possible types

- *bool* – True: defaults to ‘b’ False: Don’t draw any colorbar
- *str* – The string can be a combination of one of the following strings: {‘fr’, ‘fb’, ‘fl’, ‘ft’, ‘b’, ‘r’, ‘sv’, ‘sh’}
  - ‘b’, ‘r’ stand for bottom and right of the axes
  - ‘fr’, ‘fb’, ‘fl’, ‘ft’ stand for bottom, right, left and top of the figure
  - ‘sv’ and ‘sh’ stand for a vertical or horizontal colorbar in a separate figure

- *list* – A containing one of the above positions

Examples

Draw a colorbar at the bottom and left of the axes:

```
>>> plotter.update(cbar='bl')
```

cticks

Specify the tick locations of the colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  bounds let the `bounds` keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`

maskbetween

Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

`maskless`, `maskleq`, `maskgreater`, `maskgeq`

maskgeq
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

`maskless`, `maskleq`, `maskgreater`, `maskbetween`

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

`maskless`, `maskleq`, `maskgeq`, `maskbetween`

maskleq
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

`maskless`, `maskgreater`, `maskgeq`, `maskbetween`

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

`maskleq`, `maskgreater`, `maskgeq`, `maskbetween`

post
Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *`post_timing`* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

`post_timing` Determine the timing of this formatoption

`post_timing`
Determine when to run the *`post`* formatoption

This formatoption determines, whether the *`post`* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

`post` The post processing formatoption

class `psy_simple.plotters.`**`Simple2DBase`**(*data=None, ax=None, auto_update=None, project=None, draw=None, make_plot=True, clear=False, enable_post=False, **kwargs*)

Bases: *`psy_simple.plotters.Base2D`*

Base class for *Simple2DPlotter* and `psyplot.plotter.maps.FieldPlotter` that defines the data management

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the *post* formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Attributes

allowed_dims	The number of allowed dimensions in the for the visualization.

Methods

check_data(name, dims, is_unstructured)	A validation method for the data shape

Color coding formatoptions

miss_color	Set the color for missing values
bounds	Specify the boundaries of the colorbar
cbar	Specify the position of the colorbars
cbarspacing	Specify the spacing of the bounds in the colorbar
cmap	Specify the color map
ctickprops	Specify the font properties of the colorbar ticklabels
cticksize	Specify the font size of the colorbar ticklabels
ctickweight	Specify the fontweight of the colorbar ticklabels
extend	Draw arrows at the side of the colorbar

Label formatoptions

clabel	Show the colorbar label
clabelprops	Properties of the Colorbar label

Continued on next page

Table 222 – continued from previous page

`clabelsize`	Set the size of the Colorbar label
`clabelweight`	Set the fontweight of the Colorbar label

Axis tick formatoptions

`cticklabels`	Specify the colorbar ticklabels
`cticks`	Specify the tick locations of the colorbar

Post processing formatoptions

`post`	Apply your own postprocessing script
`post_timing`	Determine when to run the `post` formatoption

Miscallaneous formatoptions

`datagrid`	Show the grid of the data

allowed_dims = 2
The number of allowed dimensions in the for the visualization. If the array is unstructured, one dimension will be subtracted

classmethod check_data(*name*, *dims*, *is_unstructured*)
A validation method for the data shape

Parameters

- **name** (`str or list of str`) – The variable names (one variable per array)
- **dims** (`list with length 1 or list of lists with length 1`) – The dimension of the arrays. Only 1D-Arrays are allowed
- **is_unstructured** (`bool or list of bool`) – True if the corresponding array is unstructured.

Returns

- *list of bool or None* – True, if everything is okay, False in case of a serious error, None if it is intermediate. Each object in this list corresponds to one in the given *name*
- *list of str* – The message giving more information on the reason. Each object in this list corresponds to one in the given *name*

miss_color
Set the color for missing values

Possible types

- *None* – Use the default from the colormap
- *string, tuple.* – Defines the color of the grid.

bounds
Specify the boundaries of the colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

`cmap` Specifies the colormap

`cbar`
Specify the position of the colorbars

Possible types

- *bool* – True: defaults to 'b' False: Don't draw any colorbar

- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

Examples

Draw a colorbar at the bottom and left of the axes:

```
>>> plotter.update(cbar='bl')
```

cbarspacing

Specify the spacing of the bounds in the colorbar

Possible types

str {'uniform', 'proportional'} – if `'uniform'`, every color has exactly the same width in the colorbar, if `'proportional'`, the size is chosen according to the data

cmap

Specify the color map

This formatoption specifies the color coding of the data via a `matplotlib.colors.Colormap`

Possible types

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

bounds specifies the boundaries of the colormap

ctickprops

Specify the font properties of the colorbar ticklabels

Possible types

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

cticksize, *ctickweight*, *cticklabels*, *cticks*, `vcticksize`, `vctickweight`, `vcticklabels`, `vcticks`

cticksize

Specify the font size of the colorbar ticklabels

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

ctickweight, *ctickprops*, *cticklabels*, *cticks*, vctickweight, vctickprops, vcticklabels, vcticks

ctickweight

Specify the fontweight of the colorbar ticklabels

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

cticksize, *ctickprops*, *cticklabels*, *cticks*, vcticksize, vctickprops, vcticklabels, vcticks

extend

Draw arrows at the side of the colorbar

Possible types

str {'neither', 'both', 'min' or 'max'} – If not 'neither', make pointed end(s) for out-of-range values

clabel

Show the colorbar label

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`

  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `set_label()` method.

See also:

clabelsize, *clabelweight*, *clabelprops*

clabelprops
Properties of the Colorbar label

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

clabel, *clabelsize*, *clabelweight*

clabelsize
Set the size of the Colorbar label

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

clabel, *clabelweight*, *clabelprops*

clabelweight
Set the fontweight of the Colorbar label

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

clabel, *clabelsize*, *clabelprops*

cticklabels
Specify the colorbar ticklabels

Possible types

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

cticks, *cticksize*, *ctickweight*, *ctickprops*, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

cticks
Specify the tick locations of the colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  bounds let the *bounds* keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

cticklabels

post
Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

datagrid

Show the grid of the data

This formatoption shows the grid of the data (without labels)

Possible types

- *None* – Don't show the data grid
- *str* – A linestyle in the form `'k-'`, where `'k'` is the color and `'-'` the linestyle.
- *dict* – any keyword arguments that are passed to the plotting function (`matplotlib.pyplot.triplot()` for triangular grids and `matplotlib.pyplot.hlines()` for rectilinear grids)

class psy_simple.plotters.**Simple2DPlotter**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: *psy_simple.plotters.Simple2DBase*, *psy_simple.plotters.SimplePlotterBase*

Plotter for visualizing 2-dimensional data.

See also:

`psyplot.plotter.maps.FieldPlotter`

Miscallaneous formatoptions

interp_bounds	Interpolate grid cell boundaries for 2D plots
datagrid	Show the grid of the data
sym_lims	Make x- and y-axis symmetric
ticksize	Change the ticksize of the ticklabels
tickweight	Change the fontweight of the ticks

Color coding formatoptions

levels	The levels for the contour plot
bounds	Specify the boundaries of the colorbar
cbar	Specify the position of the colorbars
cbarspacing	Specify the spacing of the bounds in the colorbar
cmap	Specify the color map
ctickprops	Specify the font properties of the colorbar ticklabels
cticksize	Specify the font size of the colorbar ticklabels
ctickweight	Specify the fontweight of the colorbar ticklabels
extend	Draw arrows at the side of the colorbar
miss_color	Set the color for missing values

Plot formatoptions

plot	Specify the plotting method

Axes formatoptions

`transpose`	Switch x- and y-axes
`xlim`	Set the x-axis limits
`ylim`	Set the y-axis limits
`axiscolor`	Color the x- and y-axes
`grid`	Display the grid
`tight`	Automatically adjust the plots.

Axis tick formatoptions

`xticks`	Modify the x-axis ticks
`yticks`	Modify the y-axis ticks
`cticklabels`	Specify the colorbar ticklabels
`cticks`	Specify the tick locations of the colorbar
`xrotation`	Rotate the x-axis ticks
`xticklabels`	Modify the x-axis ticklabels
`xtickprops`	Specify the x-axis tick parameters
`yrotation`	Rotate the y-axis ticks
`yticklabels`	Modify the y-axis ticklabels
`ytickprops`	Specify the y-axis tick parameters

Masking formatoptions

`maskbetween`	Mask data points between two numbers
`maskgeq`	Mask data points greater than or equal to a number
`maskgreater`	Mask data points greater than a number
`maskleq`	Mask data points smaller than or equal to a number
`maskless`	Mask data points smaller than a number

Label formatoptions

`clabel`	Show the colorbar label
`clabelprops`	Properties of the Colorbar label
`clabelsize`	Set the size of the Colorbar label
`clabelweight`	Set the fontweight of the Colorbar label
`figtitle`	Plot a figure title
`figtitleprops`	Properties of the figure title
`figtitlesize`	Set the size of the figure title
`figtitleweight`	Set the fontweight of the figure title
`labelprops`	Set the font properties of both, x- and y-label
`labelsize`	Set the size of both, x- and y-label
`labelweight`	Set the font size of both, x- and y-label
`text`	Add text anywhere on the plot
`title`	Show the title
`titleprops`	Properties of the title
`titlesize`	Set the size of the title
`titleweight`	Set the fontweight of the title
`xlabel`	Set the x-axis label
`ylabel`	Set the y-axis label

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the *post* formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

color = None

interp_bounds

Interpolate grid cell boundaries for 2D plots

This formatoption can be used to tell enable and disable the interpolation of grid cell boundaries. Usually, netCDF files only contain the centered coordinates. In this case, we interpolate the boundaries between the grid cell centers.

Possible types

- *None* – Interpolate the boundaries, except for circumpolar grids
- *bool* – If True (the default), the grid cell boundaries are inter- and extrapolated. Otherwise, if False, the coordinate centers are used and the default behaviour of matplotlib cuts of the most outer row and column of the 2D-data. Note that this results in a slight shift of the data

legend = None

legendlabels = None

levels

The levels for the contour plot

This formatoption sets the levels for the filled contour plot and only has an effect if the *plot* Formatoption is set to `'contourf'`

Possible types

- *None* – Use the settings from the `bounds` formatoption and if this does not specify boundaries, use 11
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.

plot
Specify the plotting method

Possible types

- *None* – Don't make any plotting
- *'mesh'* – Use the `matplotlib.pyplot.pcolormesh()` function to make the plot

transpose
Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

xlim
Set the x-axis limits

Possible types

- *None* – To not change the current limits

- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

ylim

xticks

Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

`xticklabels`, `ticksize`, `tickweight`, `xtickprops`, `yticks`

ylim

Set the y-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

xlim

yticks

Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

yticklabels, *ticksize*, *tickweight*, *ytickprops*

xticks for possible examples

bounds

Specify the boundaries of the colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

---

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

`cmap` Specifies the colormap

`cbar`
Specify the position of the colorbars

Possible types

- *bool* – True: defaults to 'b' False: Don't draw any colorbar

- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

Examples

Draw a colorbar at the bottom and left of the axes:

```
>>> plotter.update(cbar='bl')
```

cbarspacing

Specify the spacing of the bounds in the colorbar

Possible types

str {'uniform', 'proportional'} – if `'uniform'`, every color has exactly the same width in the colorbar, if `'proportional'`, the size is chosen according to the data

cmap

Specify the color map

This formatoption specifies the color coding of the data via a `matplotlib.colors.Colormap`

Possible types

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

bounds specifies the boundaries of the colormap

ctickprops

Specify the font properties of the colorbar ticklabels

Possible types

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`cticksize`, *`ctickweight`*, *`cticklabels`*, *`cticks`*, `vcticksize`, `vctickweight`, `vcticklabels`, `vcticks`

cticksize
Specify the font size of the colorbar ticklabels

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

ctickweight, *ctickprops*, *cticklabels*, *cticks*, vctickweight, vctickprops, vcticklabels, vcticks

ctickweight
Specify the fontweight of the colorbar ticklabels

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

cticksize, *ctickprops*, *cticklabels*, *cticks*, vcticksize, vctickprops, vcticklabels, vcticks

extend
Draw arrows at the side of the colorbar

Possible types

str {'neither', 'both', 'min' or 'max'} – If not 'neither', make pointed end(s) for out-of-range values

miss_color
Set the color for missing values

Possible types

- *None* – Use the default from the colormap
- *string, tuple.* – Defines the color of the grid.

maskbetween
Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskgeq*

maskgeq

Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater

Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgeq*, *maskbetween*

maskleq

Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless

Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

clabel

Show the colorbar label

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)

- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `set_label()` method.

See also:

clabelsize, *clabelweight*, *clabelprops*

clabelprops
Properties of the Colorbar label

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

clabel, *clabelsize*, *clabelweight*

clabelsize
Set the size of the Colorbar label

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

clabel, *clabelweight*, *clabelprops*

clabelweight
Set the fontweight of the Colorbar label

Possible types

- *float* – a float between 0 and 1000

- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

clabel, *clabelsize*, *clabelprops*

figtitle
Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this `Figtitle` instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *title* formatoption.

See also:

title, *figtitlesize*, *figtitleweight*, *figtitleprops*

figtitleprops
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

figtitle, *figtitlesize*, *figtitleweight*

figtitlesize
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

figtitle, *figtitleweight*, *figtitleprops*

figtitleweight
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

figtitle, *figtitlesize*, *figtitleprops*

labelprops
Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

xlabel, *ylabel*, *labelsize*, *labelweight*

labelsize
Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`xlabel`, `ylabel`, `labelweight`, `labelprops`

`labelweight`
Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`xlabel`, `ylabel`, `labelsize`, `labelprops`

`text`
Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

title, *figtitle*

title

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the *figtitle* formatoption.

See also:

figtitle, *titlesize*, *titleweight*, *titleprops*

titleprops
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

title, *titlesize*, *titleweight*

titlesize
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

title, *titleweight*, *titleprops*

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

title, *titlesize*, *titleprops*

xlabel
Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`

- desc: `%(long_name)s [%(units)s]`
- dinfo: `%B %d, %Y`
- tinfo: `%H:%M`
- sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

ylabel

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

axiscolor

Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

grid

Display the grid

Show the grid on the plot with the specified color.

Possible types

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

tight

Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

Warning: There is no update method to undo what happend after this formatoption is set to True!

cticklabels

Specify the colorbar ticklabels

Possible types

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

cticks, *cticksize*, *ctickweight*, *ctickprops*, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

cticks

Specify the tick locations of the colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually

- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  bounds let the `bounds` keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`

xrotation
Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`yrotation`

xticklabels
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

xtickprops

Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *ytickprops*

yrotation

Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

xrotation

yticklabels

Modify the y-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

yticks, *ticksize*, *tickweight*, *ytickprops*, *xticklabels*

ytickprops

Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *xtickprops*

datagrid
Show the grid of the data

This formatoption shows the grid of the data (without labels)

Possible types

- *None* – Don't show the data grid
- *str* – A linestyle in the form `'k-'`, where `'k'` is the color and `'-'` the linestyle.
- *dict* – any keyword arguments that are passed to the plotting function (`matplotlib.pyplot.triplot()` for triangular grids and `matplotlib.pyplot.hlines()` for rectilinear grids)

sym_lims
Make x- and y-axis symmetric

Possible types

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

ticksize
Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

tickweight, *xtickprops*, *ytickprops*

tickweight
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

ticksize, *xtickprops*, *ytickprops*

class psy_simple.plotters.**SimplePlot2D**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.Plot2D*

Specify the plotting method

Possible types

Attributes

array	The (masked) data array that is plotted
bounds	bounds Formatoption instance in the plotter
cmap	cmap Formatoption instance in the plotter
data_dependent	bool(x) -> bool
dependencies	list() -> new empty list
interp_bounds	interp_bounds Formatoption instance in the plotter
levels	levels Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter
triangles	The `matplotlib.tri.Triangulation` instance containing the
xbounds	Boundaries of the x-coordinate
xcoord	The x coordinate `xarray.Variable`
ybounds	Boundaries of the y-coordinate
ycoord	The y coordinate `xarray.Variable`

- *None* – Don't make any plotting
- *'mesh'* – Use the `matplotlib.pyplot.pcolormesh()` function to make the plot

array
The (masked) data array that is plotted

bounds
bounds Formatoption instance in the plotter

cmap
cmap Formatoption instance in the plotter

data_dependent = True

dependencies = ['levels', 'interp_bounds', 'transpose']

interp_bounds
interp_bounds Formatoption instance in the plotter

levels
levels Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

triangles
The `matplotlib.tri.Triangulation` instance containing the spatial informations

xbounds
Boundaries of the x-coordinate

xcoord
The x coordinate `xarray.Variable`

ybounds
Boundaries of the y-coordinate

ycoord
The y coordinate `xarray.Variable`

class psy_simple.plotters.**SimplePlotterBase**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)
Bases: *psy_simple.base.BasePlotter*, *psy_simple.plotters.XYTickPlotter*

Base class for all simple plotters

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw''* parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the *post* formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Attributes

allowed_dims	The number of allowed dimensions in the for the visualization.
allowed_vars	The number variables that one data array visualized by this plotter might have.

Axes formatoptions

axiscolor	Color the x- and y-axes
grid	Display the grid
transpose	Switch x- and y-axes
xlim	Set the x-axis limits
ylim	Set the y-axis limits
tight	Automatically adjust the plots.

Methods

check_data(name, dims[, is_unstructured])	A validation method for the data shape

Color coding formatoptions

color	Set the color coding

Miscallaneous formatoptions

legend	Draw a legend
legendlabels	Set the labels of the arrays in the legend
sym_lims	Make x- and y-axis symmetric
ticksize	Change the ticksize of the ticklabels
tickweight	Change the fontweight of the ticks

Masking formatoptions

maskbetween	Mask data points between two numbers
maskgeq	Mask data points greater than or equal to a number
maskgreater	Mask data points greater than a number
maskleq	Mask data points smaller than or equal to a number
maskless	Mask data points smaller than a number

Label formatoptions

figtitle	Plot a figure title
figtitleprops	Properties of the figure title
figtitlesize	Set the size of the figure title
figtitleweight	Set the fontweight of the figure title
labelprops	Set the font properties of both, x- and y-label
labelsize	Set the size of both, x- and y-label
labelweight	Set the font size of both, x- and y-label

Continued on next page

Table 241 – continued from previous page

text	Add text anywhere on the plot
title	Show the title
titleprops	Properties of the title
titlesize	Set the size of the title
titleweight	Set the fontweight of the title
xlabel	Set the x-axis label
ylabel	Set the y-axis label

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

Axis tick formatoptions

xrotation	Rotate the x-axis ticks
xticklabels	Modify the x-axis ticklabels
xtickprops	Specify the x-axis tick parameters
xticks	Modify the x-axis ticks
yrotation	Rotate the y-axis ticks
yticklabels	Modify the y-axis ticklabels
ytickprops	Specify the y-axis tick parameters
yticks	Modify the y-axis ticks

allowed_dims = 1
The number of allowed dimensions in the for the visualization. If the array is unstructured, one dimension will be subtracted

allowed_vars = 1
The number variables that one data array visualized by this plotter might have.

axiscolor
Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

classmethod check_data(*name*, *dims*, *is_unstructured=None*)

A validation method for the data shape

Parameters

- **name** (`str or list of str`) – The variable names (at maximum `allowed_vars` variables per array)
- **dims** (`list with length 1 or list of lists with length 1`) – The dimension of the arrays. Only 1D-Arrays are allowed
- **is_unstructured** (`bool or list of bool, optional`) – True if the corresponding array is unstructured. This keyword is ignored

Returns

- *list of bool or None* – True, if everything is okay, False in case of a serious error, None if it is intermediate. Each object in this list corresponds to one in the given *name*
- *list of str* – The message giving more information on the reason. Each object in this list corresponds to one in the given *name*

color

Set the color coding

This formatoptions sets the color of the lines, bars, etc.

Possible types

- *None* – to use the axes color_cycle
- *iterable* – (e.g. list) to specify the colors manually
- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.ColorMap* – to automatically choose the colors according to the number of lines, etc. from the given colormap

grid

Display the grid

Show the grid on the plot with the specified color.

Possible types

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

legend

Draw a legend

This formatoption determines where and if to draw the legend. It uses the `labels` formatoption to determine the labels.

Possible types

- *bool* – Draw a legend or not
- *str or int* – Specifies where to plot the legend (i.e. the location)
- *dict* – Give the keywords for the `matplotlib.pyplot.legend()` function

See also:

`labels`

legendlabels

Set the labels of the arrays in the legend

This formatoption specifies the labels for each array in the legend. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)

- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – A single string that shall be used for all arrays.
- *list of str* – Same as a single string but specified for each array

See also:

legend

sym_lims
Make x- and y-axis symmetric

Possible types

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

transpose
Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

xlim
Set the x-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

`ylim`
Set the y-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`xlim`

`maskbetween`
Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskgeq*

maskgeq
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgeq*, *maskbetween*

maskleq
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

figtitle
Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this `Figtitle` instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *`title`* formatoption.

See also:

`title`, *`figtitlesize`*, *`figtitleweight`*, *`figtitleprops`*

`figtitleprops`
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`figtitle`, *`figtitlesize`*, *`figtitleweight`*

`figtitlesize`
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

figtitle, *figtitleweight*, *figtitleprops*

figtitleweight
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

figtitle, *figtitlesize*, *figtitleprops*

labelprops
Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

xlabel, *ylabel*, *labelsize*, *labelweight*

labelsize
Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

xlabel, *ylabel*, *labelweight*, *labelprops*

labelweight

Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

xlabel, *ylabel*, *labelsize*, *labelprops*

text

Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

title, *figtitle*

title
Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the *figtitle* formatoption.

See also:

figtitle, *titlesize*, *titleweight*, *titleprops*

titleprops
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

title, *titlesize*, *titleweight*

titlesize
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

title, *titleweight*, *titleprops*

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

title, *titlesize*, *titleprops*

xlabel
Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

ylabel

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

tight

Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

> **Warning:** There is no update method to undo what happend after this formatoption is set to True!

xrotation

Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`yrotation`

`xticklabels`
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

`xtickprops`
Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`xticks`, `yticks`, `ticksize`, `tickweight`, `ytickprops`

`xticks`
Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks

- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

`xticklabels`, `ticksize`, `tickweight`, `xtickprops`, `yticks`

yrotation
Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`xrotation`

yticklabels
Modify the y-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`yticks`, `ticksize`, `tickweight`, `ytickprops`, `xticklabels`

ytickprops
Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`xticks`, `yticks`, `ticksize`, `tickweight`, `xtickprops`

yticks
Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks

- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

`yticklabels`, `ticksize`, `tickweight`, `ytickprops`

`xticks` for possible examples

`ticksize`

Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`tickweight`, `xtickprops`, `ytickprops`

tickweight
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

ticksize, *xtickprops*, *ytickprops*

class psy_simple.plotters.**SimpleVectorPlot**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.VectorPlot*

Choose the vector plot type

Possible types

Attributes

arrowsize	arrowsize Formatoption instance in the plotter
arrowstyle	arrowstyle Formatoption instance in the plotter
bounds	bounds Formatoption instance in the plotter
cmap	cmap Formatoption instance in the plotter
color	color Formatoption instance in the plotter
data_dependent	bool(x) -> bool
density	density Formatoption instance in the plotter
linewidth	linewidth Formatoption instance in the plotter
transform	transform Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter

Methods

set_value(value, *args, **kwargs)	Set (and validate) the value in the plotter.

str – Plot types can be either

quiver to make a quiver plot

stream to make a stream plot

arrowsize
arrowsize Formatoption instance in the plotter

arrowstyle
arrowstyle Formatoption instance in the plotter

bounds
bounds Formatoption instance in the plotter

cmap
cmap Formatoption instance in the plotter

color
color Formatoption instance in the plotter

data_dependent = True

density
density Formatoption instance in the plotter

linewidth
linewidth Formatoption instance in the plotter

set_value(*value*, **args*, ***kwargs*)
Set (and validate) the value in the plotter. This method is called by the plotter when it attempts to change the value of the formatoption.

Parameters

- **value** – Value to set
- **validate** (*bool*) – if True, validate the *value* before it is set
- **todefault** (*bool*) – True if the value is updated to the default value

transform
transform Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

class psy_simple.plotters.**SimpleVectorPlotter**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: *psy_simple.plotters.BaseVectorPlotter*, *psy_simple.plotters.SimplePlotterBase*

Plotter for visualizing 2-dimensional vector data

See also:

psyplot.plotter.maps.VectorPlotter

Plot formatoptions

plot	Choose the vector plot type

Axes formatoptions

xlim	Set the x-axis limits
ylim	Set the y-axis limits
axiscolor	Color the x- and y-axes
grid	Display the grid
tight	Automatically adjust the plots.
transpose	Switch x- and y-axes

Axis tick formatoptions

`xticks`	Modify the x-axis ticks
`yticks`	Modify the y-axis ticks
`cticklabels`	Specify the colorbar ticklabels
`cticks`	Specify the tick locations of the vector colorbar
`xrotation`	Rotate the x-axis ticks
`xticklabels`	Modify the x-axis ticklabels
`xtickprops`	Specify the x-axis tick parameters
`yrotation`	Rotate the y-axis ticks
`yticklabels`	Modify the y-axis ticklabels
`ytickprops`	Specify the y-axis tick parameters

Color coding formatoptions

`bounds`	Specify the boundaries of the vector colorbar
`cbar`	Specify the position of the vector plot colorbars
`cbarspacing`	Specify the spacing of the bounds in the colorbar
`cmap`	Specify the color map
`color`	Set the color for the arrows
`ctickprops`	Specify the font properties of the colorbar ticklabels
`cticksize`	Specify the font size of the colorbar ticklabels
`ctickweight`	Specify the fontweight of the colorbar ticklabels
`extend`	Draw arrows at the side of the colorbar

Masking formatoptions

`maskbetween`	Mask data points between two numbers
`maskgeq`	Mask data points greater than or equal to a number
`maskgreater`	Mask data points greater than a number
`maskleq`	Mask data points smaller than or equal to a number
`maskless`	Mask data points smaller than a number

Label formatoptions

`clabel`	Show the colorbar label
`clabelprops`	Properties of the Colorbar label
`clabelsize`	Set the size of the Colorbar label
`clabelweight`	Set the fontweight of the Colorbar label
`figtitle`	Plot a figure title
`figtitleprops`	Properties of the figure title
`figtitlesize`	Set the size of the figure title
`figtitleweight`	Set the fontweight of the figure title
`labelprops`	Set the font properties of both, x- and y-label
`labelsize`	Set the size of both, x- and y-label
`labelweight`	Set the font size of both, x- and y-label
`text`	Add text anywhere on the plot
`title`	Show the title
`titleprops`	Properties of the title
`titlesize`	Set the size of the title

Continued on next page

Table 251 – continued from previous page

titleweight	Set the fontweight of the title
xlabel	Set the x-axis label
ylabel	Set the y-axis label

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

Vector plot formatoptions

arrowsize	Change the size of the arrows
arrowstyle	Change the style of the arrows
density	Change the density of the arrows

Miscallaneous formatoptions

datagrid	Show the grid of the data
linewidth	Change the linewidth of the arrows
sym_lims	Make x- and y-axis symmetric
ticksize	Change the ticksize of the ticklabels
tickweight	Change the fontweight of the ticks

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the *post* formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

legend = None

legendlabels = None

plot
Choose the vector plot type

Possible types

str – Plot types can be either

quiver to make a quiver plot

stream to make a stream plot

xlim
Set the x-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

xticks
Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually

- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

`xticklabels`, `ticksize`, `tickweight`, `xtickprops`, `yticks`

ylim

Set the y-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

xlim

yticks

Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

hour draw ticks every hour

day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

yticklabels, *ticksize*, *tickweight*, *ytickprops*

xticks for possible examples

bounds

Specify the boundaries of the vector colorbar

Possible types

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

cmap Specifies the colormap

cbar
Specify the position of the vector plot colorbars

Possible types

- *bool* – True: defaults to 'b' False: Don't draw any colorbar
- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

cbarspacing
Specify the spacing of the bounds in the colorbar

Possible types

str {'uniform', 'proportional'} – if `'uniform'`, every color has exactly the same width in the colorbar, if `'proportional'`, the size is chosen according to the data

cmap
Specify the color map

This formatoption specifies the color coding of the data via a `matplotlib.colors.Colormap`

Possible types

- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.Colormap* – The colormap instance to use

See also:

bounds specifies the boundaries of the colormap

color
Set the color for the arrows

This formatoption can be used to set a single color for the vectors or define the color coding

Possible types

- *float* – Determines the greyness
- *color* – Defines the same color for all arrows. The string can be either a html hex string (e.g. '#eeefff'), a single letter (e.g. 'b': blue, 'g': green, 'r': red, 'c': cyan, 'm': magenta, 'y': yellow, 'k': black, 'w': white) or any other color
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *2D-array* – The values determine the color for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

arrowsize, *arrowstyle*, *density*, *linewidth*

ctickprops
Specify the font properties of the colorbar ticklabels

Possible types

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

cticksize, *ctickweight*, *cticklabels*, *cticks*, vcticksize, vctickweight, vcticklabels, vcticks

cticksize
Specify the font size of the colorbar ticklabels

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

ctickweight, *ctickprops*, *cticklabels*, *cticks*, vctickweight, vctickprops, vcticklabels, vcticks

ctickweight
Specify the fontweight of the colorbar ticklabels

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

cticksize, *ctickprops*, *cticklabels*, *cticks*, `vcticksize`, `vctickprops`, `vcticklabels`, `vcticks`

extend
Draw arrows at the side of the colorbar

Possible types

str {'neither', 'both', 'min' or 'max'} – If not 'neither', make pointed end(s) for out-of-range values

maskbetween
Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskgeq*

maskgeq
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgeq*, *maskbetween*

maskleq
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

clabel
Show the colorbar label

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `set_label()` method.

See also:

clabelsize, *clabelweight*, *clabelprops*

clabelprops
Properties of the Colorbar label

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`clabel`, `clabelsize`, `clabelweight`

clabelsize
Set the size of the Colorbar label

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`clabel`, `clabelweight`, `clabelprops`

clabelweight
Set the fontweight of the Colorbar label

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`clabel`, `clabelsize`, `clabelprops`

figtitle
Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`

– sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this `Figtitle` instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *`title`* formatoption.

See also:

`title`, *`figtitlesize`*, *`figtitleweight`*, *`figtitleprops`*

`figtitleprops`
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`figtitle`, *`figtitlesize`*, *`figtitleweight`*

`figtitlesize`
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`figtitle`, *`figtitleweight`*, *`figtitleprops`*

`figtitleweight`
Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

figtitle, *figtitlesize*, *figtitleprops*

labelprops
Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

xlabel, *ylabel*, *labelsize*, *labelweight*

labelsize
Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

xlabel, *ylabel*, *labelweight*, *labelprops*

labelweight
Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

xlabel, *ylabel*, *labelsize*, *labelprops*

text
Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

title, *figtitle*

title

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are

- dtinfo: `%B %d, %Y. %H:%M`
- desc: `%(long_name)s [%(units)s]`
- dinfo: `%B %d, %Y`
- tinfo: `%H:%M`
- sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the *figtitle* formatoption.

See also:

figtitle, *titlesize*, *titleweight*, *titleprops*

titleprops
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

title, *titlesize*, *titleweight*

titlesize
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

title, *titleweight*, *titleprops*

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

title, *titlesize*, *titleprops*

xlabel

Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

ylabel

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).

- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

axiscolor

Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

grid

Display the grid

Show the grid on the plot with the specified color.

Possible types

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

tight

Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

> **Warning:** There is no update method to undo what happend after this formatoption is set to True!

transpose

Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

cticklabels
Specify the colorbar ticklabels

Possible types

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`cticks`, *`cticksize`*, *`ctickweight`*, *`ctickprops`*, `vcticks`, `vcticksize`, `vctickweight`, `vctickprops`

cticks
Specify the tick locations of the vector colorbar

Possible types

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, …]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  bounds let the *`bounds`* keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)
- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`, `vcticklabels`

xrotation
Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`yrotation`

xticklabels
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

xtickprops
Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`xticks`, `yticks`, `ticksize`, `tickweight`, `ytickprops`

yrotation
Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

`xrotation`

yticklabels
Modify the y-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

yticks, *ticksize*, *tickweight*, *ytickprops*, *xticklabels*

ytickprops
Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *xtickprops*

arrowsize
Change the size of the arrows

Possible types

- *None* – make no scaling
- *float* – Factor scaling the size of the arrows

See also:

arrowstyle, *linewidth*, *density*, *color*

arrowstyle
Change the style of the arrows

Possible types

str – Any arrow style string (see `FancyArrowPatch`)

Notes

This formatoption only has an effect for stream plots

See also:

arrowsize, *linewidth*, *density*, *color*

density
Change the density of the arrows

Possible types

- *float* – Scales the density of the arrows in x- and y-direction (1.0 means no scaling)
- *tuple (x, y)* – Defines the scaling in x- and y-direction manually

Notes

quiver plots do not support density scaling

datagrid
Show the grid of the data

This formatoption shows the grid of the data (without labels)

Possible types

- *None* – Don't show the data grid
- *str* – A linestyle in the form `'k-'`, where `'k'` is the color and `'-'` the linestyle.
- *dict* – any keyword arguments that are passed to the plotting function (`matplotlib.pyplot.triplot()` for triangular grids and `matplotlib.pyplot.hlines()` for rectilinear grids)

linewidth
Change the linewidth of the arrows

Possible types

- *float* – give the linewidth explicitly
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *tuple (string, float)* – *string* may be one of the above strings, *float* may be a scaling factor
- *2D-array* – The values determine the linewidth for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

arrowsize, *arrowstyle*, *density*, *color*

sym_lims
Make x- and y-axis symmetric

Possible types

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

ticksize
Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

tickweight, *xtickprops*, *ytickprops*

tickweight
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

ticksize, *xtickprops*, *ytickprops*

class `psy_simple.plotters.`**SymmetricLimits**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Make x- and y-axis symmetric

Possible types

Attributes

dependencies	list() -> new empty list
name	str(object='') -> str
xlim	xlim Formatoption instance in the plotter
ylim	ylim Formatoption instance in the plotter

Methods

update(value)	Method that is call to update the formatoption on the axes

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, *dependencies* and `connections` attributes, with values being the name of the new formatoption in this plotter.

dependencies = ['xlim', 'ylim']

name = 'Symmetric x- and y-axis limits'

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

xlim
xlim Formatoption instance in the plotter

ylim
ylim Formatoption instance in the plotter

class psy_simple.plotters.**TickLabels**(**args, **kwargs*)
Bases: `psy_simple.plotters.TickLabelsBase`, `psy_simple.plotters.TicksManager`

Methods

`set_default_formatters`([which])	Sets the default formatters that is used for updating to None
`set_formatter`(formatter[, which])	Sets a given formatter
`update`(value)	Method that is call to update the formatoption on the axes

Attributes

`transpose`	transpose Formatoption instance in the plotter

set_default_formatters(*which=None*)
Sets the default formatters that is used for updating to None

Parameters **which** (`{None, 'minor', 'major'}`) – Specify which locator shall be set

set_formatter(*formatter, which=None*)
Sets a given formatter

transpose
transpose Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**TickLabelsBase**(**args, **kwargs*)
Bases: `psy_simple.plotters.TicksManagerBase`

Abstract base class for ticklabels

Possible types

Attributes

`axis`	The axis on the axes to modify the ticks of
`dependencies`	list() -> new empty list
`group`	str(object='') -> str
`transpose`	transpose Formatoption instance in the plotter

Methods

`initialize_plot`(value)	Method that is called when the plot is made the first time
`set_default_formatters`()	Sets the default formatters that is used for updating to None
`set_formatter`(formatter)	Sets a given formatter
`set_stringformatter`(s)	

Continued on next page

Table 260 – continued from previous page

set_ticklabels(labels)	Sets the given tick labels
update_axis(value)	

- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

axis
The axis on the axes to modify the ticks of

dependencies = ['transpose']

group = 'ticks'

initialize_plot(*value*)
Method that is called when the plot is made the first time

Parameters **value** – The value to use for the initialization

set_default_formatters()
Sets the default formatters that is used for updating to None

set_formatter(*formatter*)
Sets a given formatter

set_stringformatter(*s*)

set_ticklabels(*labels*)
Sets the given tick labels

transpose
transpose Formatoption instance in the plotter

update_axis(*value*)

class psy_simple.plotters.**TickPropsBase**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: *psy_simple.plotters.TicksManagerBase*

Abstract base class for tick parameters

Possible types

Attributes

axisname	The name of the axis (either 'x' or 'y')

Methods

update_axis(value)	

dict – Items may be anything of the `matplotlib.pyplot.tick_params()` function

Parameters

- **key** (*str*) – formatoption key in the *plotter*

- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

axisname

The name of the axis (either 'x' or 'y')

update_axis(*value*)

class psy_simple.plotters.**TickSize**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.TickSizeBase*, *psy_simple.plotters.TicksOptions*, `psyplot.plotter.DictFormatoption`

Change the ticksize of the ticklabels

Possible types

Attributes

dependencies	list() -> new empty list
name	str(object='') -> str
xtickprops	xtickprops Formatoption instance in the plotter
ytickprops	ytickprops Formatoption instance in the plotter

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`tickweight`, *xtickprops*, *ytickprops*

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.

- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, *dependencies* and `connections` attributes, with values being the name of the new formatoption in this plotter.

dependencies = ['xtickprops', 'ytickprops']

name = 'Font size of the ticklabels'

xtickprops
xtickprops Formatoption instance in the plotter

ytickprops
ytickprops Formatoption instance in the plotter

class psy_simple.plotters.**TickSizeBase**(*key, plotter=None, index_in_list=None, additional_children=[], additional_dependencies=[], **kwargs*)
Bases: *psy_simple.plotters.TicksOptions*

Abstract base class for modifying tick sizes

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

Methods

update_axis(value)	

update_axis(*value*)

class psy_simple.plotters.**TickWeight**(*key, plotter=None, index_in_list=None, additional_children=[], additional_dependencies=[], **kwargs*)
Bases: *psy_simple.plotters.TickWeightBase*, *psy_simple.plotters.TicksOptions*,

psyplot.plotter.DictFormatoption

Change the fontweight of the ticks

Possible types

Attributes

dependencies	list() -> new empty list
name	str(object='') -> str
xtickprops	xtickprops Formatoption instance in the plotter
ytickprops	ytickprops Formatoption instance in the plotter

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`ticksize`, *xtickprops*, *ytickprops*

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the *dependencies* attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, *dependencies* and `connections` attributes, with values being the name of the new formatoption in this plotter.

dependencies = ['xtickprops', 'ytickprops']

name = 'Font weight of the ticklabels'

xtickprops
xtickprops Formatoption instance in the plotter

ytickprops
ytickprops Formatoption instance in the plotter

class psy_simple.plotters.**TickWeightBase**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.TicksOptions*

Abstract base class for modifying font weight of ticks

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the dependencies attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the children, dependencies and connections attributes, with values being the name of the new formatoption in this plotter.

Methods

update_axis(value)	

update_axis(*value*)

class psy_simple.plotters.**TicksBase**(**args*, ***kwargs*)

Bases: *psy_simple.plotters.TicksManagerBase*, *psy_simple.plotters.DataTicksCalculator*

Abstract base class for calculating ticks

Possible types

Attributes

axis	
dependencies	list() -> new empty list
group	str(object='') -> str
plot	plot Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter

Methods

get_locator()	
initialize_plot(value)	Method that is called when the plot is made the first time
set_default_locators([which])	Sets the default locator that is used for updating to None or int

Continued on next page

Table 268 – continued from previous page

set_locator(locator)	Sets the locator corresponding of the axis
set_ticks(value)	
update_axis(value)	

- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used

axis

dependencies = ['transpose', 'plot']

get_locator()

group = 'ticks'

initialize_plot(*value*)
Method that is called when the plot is made the first time

Parameters value – The value to use for the initialization

plot
plot Formatoption instance in the plotter

set_default_locators(*which=None*)
Sets the default locator that is used for updating to None or int

Parameters which (*{None, 'minor', 'major'}*) – Specify which locator shall be set

set_locator(*locator*)
Sets the locator corresponding of the axis

Parameters

- **locator** (*matplotlib.ticker.Locator*) – The locator to set
- **which** (*{None, 'minor', 'major'}*) – Specify which locator shall be set. If None, it will be taken from the `which` attribute

set_ticks(*value*)

transpose
transpose Formatoption instance in the plotter

update_axis(*value*)

class psy_simple.plotters.**TicksManager**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)
Bases: *psy_simple.plotters.TicksManagerBase*, `psyplot.plotter.DictFormatoption`

Abstract base class for ticks formatoptions controlling major and minor ticks

This formatoption simply serves as a base that allows the simultaneous managment of major and minor ticks

Possible types

Attributes

group	str(object='') -> str

Methods

update(value)	Method that is call to update the formatoption on the axes

dict – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

group = 'ticks'

update(*value*)

Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**TicksManagerBase**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Abstract base class for formatoptions handling ticks

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords

may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

Methods

`update_axis`(val)	

update_axis(*val*)

class psy_simple.plotters.**TicksOptions**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psy_simple.plotters.TicksManagerBase`

Base class for ticklabels options that apply for x- and y-axis

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

Methods

`update`(value)	Method that is call to update the formatoption on the axes

update(*value*)

Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**Transpose**(**args*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

Methods

get_x(arr)	
get_y(arr)	
initialize_plot(value)	Method that is called when the plot is made the first time
update(value)	Method that is call to update the formatoption on the axes

Attributes

group	str(object='') -> str
name	str(object='') -> str
priority	int(x=0) -> integer

bool – If True, axes are switched

get_x(*arr*)

get_y(*arr*)

group = 'axes'

initialize_plot(*value*)
Method that is called when the plot is made the first time

Parameters **value** – The value to use for the initialization

name = 'Switch x- and y-axes'

priority = 30

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**VCLabel**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.CLabel*

Show the colorbar label of the vector plot

Set the label of the colorbar. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`

- tinfo: `%H:%M`
- sdesc: `%(name)s [%(units)s]`

Attributes

`cbar`	cbar Formatoption instance in the plotter
`plot`	plot Formatoption instance in the plotter

Possible types

str – The title for the `set_label()` method.

See also:

`vclabelsize`, `vclabelweight`, `vclabelprops`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

cbar
cbar Formatoption instance in the plotter

plot
plot Formatoption instance in the plotter

class psy_simple.plotters.**VectorBounds**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.Bounds*

Specify the boundaries of the vector colorbar

Possible types

Attributes

`array`	The numpy array of the data
`cbar`	cbar Formatoption instance in the plotter
`cmap`	cmap Formatoption instance in the plotter

Continued on next page

Table 276 – continued from previous page

`color`	color Formatoption instance in the plotter
`parents`	list() -> new empty list

Methods

`update`(*args, **kwargs)	Method that is call to update the formatoption on the axes

- *None* – make no normalization
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero
- *int* – Specifies how many ticks to use with the `'rounded'` option. I.e. if integer `i`, then this is the same as `['rounded', i]`.
- *matplotlib.colors.Normalize* – A matplotlib normalization instance

Examples

Plot 11 bounds over the whole data range:

```
>>> plotter.update(bounds='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(bounds=['minmax', 7])
```

Plot logarithmic bounds:

```
>>> from matplotlib.colors import LogNorm
>>> plotter.update(bounds=LogNorm())
```

See also:

`cmap` Specifies the colormap

array
The numpy array of the data

cbar
cbar Formatoption instance in the plotter

cmap
cmap Formatoption instance in the plotter

color
color Formatoption instance in the plotter

parents = ['color']

update(**args*, ***kwargs*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

class psy_simple.plotters.**VectorCTicks**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.CTicks*

Specify the tick locations of the vector colorbar

Possible types

Attributes

array	The numpy array of the data
bounds	bounds Formatoption instance in the plotter
cbar	cbar Formatoption instance in the plotter
color	color Formatoption instance in the plotter
dependencies	list() -> new empty list
plot	plot Formatoption instance in the plotter

- *None* – use the default ticks
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  bounds let the *bounds* keyword determine the ticks. An additional integer *i* may be specified to only use every i-th bound as a tick (see also *int* below)

- *int* – Specifies how many ticks to use with the `'bounds'` option. I.e. if integer `i`, then this is the same as `['bounds', i]`.

See also:

`cticklabels`, `vcticklabels`

array
The numpy array of the data

bounds
bounds Formatoption instance in the plotter

cbar
cbar Formatoption instance in the plotter

color
color Formatoption instance in the plotter

dependencies = ['cbar', 'bounds', 'color']

plot
plot Formatoption instance in the plotter

class psy_simple.plotters.**VectorCalculator**(**args*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Abstract formatoption that provides calculation functions for speed, etc.

Possible types

Attributes

data_dependent	bool(x) -> bool
dependencies	list() -> new empty list
plot	plot Formatoption instance in the plotter
priority	int(x=0) -> integer
transpose	transpose Formatoption instance in the plotter

string {'absolute', 'u', 'v'} – Strings may define how the formatoption is calculated. Possible strings are

- **absolute**: for the absolute wind speed
- **u**: for the u component
- **v**: for the v component

data_dependent = True

dependencies = ['plot', 'transpose']

plot
plot Formatoption instance in the plotter

priority = 20

transpose
transpose Formatoption instance in the plotter

class psy_simple.plotters.**VectorCbar**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.Cbar*

Specify the position of the vector plot colorbars

Possible types

Attributes

`bounds`	bounds Formatoption instance in the plotter
`cbarspacing`	cbarspacing Formatoption instance in the plotter
`cmap`	cmap Formatoption instance in the plotter
`color`	color Formatoption instance in the plotter
`dependencies`	list() -> new empty list
`extend`	extend Formatoption instance in the plotter
`levels`	levels Formatoption instance in the plotter
`plot`	plot Formatoption instance in the plotter
`priority`	int(x=0) -> integer

Methods

`update`(*args, **kwargs)	Updates the colorbar

- *bool* – True: defaults to 'b' False: Don't draw any colorbar
- *str* – The string can be a combination of one of the following strings: {'fr', 'fb', 'fl', 'ft', 'b', 'r', 'sv', 'sh'}
  - 'b', 'r' stand for bottom and right of the axes
  - 'fr', 'fb', 'fl', 'ft' stand for bottom, right, left and top of the figure
  - 'sv' and 'sh' stand for a vertical or horizontal colorbar in a separate figure
- *list* – A containing one of the above positions

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.
- **other_cbars** (*list of str*) – List of other colorbar formatoption keys (necessary for a sufficient resizing of the axes)

bounds
bounds Formatoption instance in the plotter

cbarspacing
cbarspacing Formatoption instance in the plotter

cmap
cmap Formatoption instance in the plotter

color
color Formatoption instance in the plotter

dependencies = ['plot', 'cmap', 'bounds', 'extend', 'cbarspacing', 'levels', 'color']

extend
extend Formatoption instance in the plotter

levels
levels Formatoption instance in the plotter

plot
plot Formatoption instance in the plotter

priority = 10

update(**args*, ***kwargs*)
Updates the colorbar

Parameters

- **value** – The value to update (see possible types)
- **no_fig_cbars** – Does not update the colorbars that are not in the axes of this plot

class psy_simple.plotters.**VectorColor**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.VectorCalculator*

Set the color for the arrows

This formatoption can be used to set a single color for the vectors or define the color coding

Possible types

Attributes

bounds	bounds Formatoption instance in the plotter
cmap	cmap Formatoption instance in the plotter
dependencies	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter

Methods

update(value)	Method that is call to update the formatoption on the axes

- *float* – Determines the greyness

- *color* – Defines the same color for all arrows. The string can be either a html hex string (e.g. '#eeefff'), a single letter (e.g. 'b': blue, 'g': green, 'r': red, 'c': cyan, 'm': magenta, 'y': yellow, 'k': black, 'w': white) or any other color
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *2D-array* – The values determine the color for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

`arrowsize`, `arrowstyle`, `density`, `linewidth`

bounds
bounds Formatoption instance in the plotter

cmap
cmap Formatoption instance in the plotter

dependencies = ['plot', 'transpose', 'cmap', 'bounds']

group = 'colors'

name = 'Color of the arrows'

plot
plot Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**VectorLineWidth**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.VectorCalculator*

Change the linewidth of the arrows

Possible types

Attributes

name	str(object='') -> str
plot	plot Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter

Methods

update(value)	Method that is call to update the formatoption on the axes

- *float* – give the linewidth explicitly
- *string {'absolute', 'u', 'v'}* – Strings may define how the formatoption is calculated. Possible strings are
  - **absolute**: for the absolute wind speed
  - **u**: for the u component
  - **v**: for the v component
- *tuple (string, float)* – *string* may be one of the above strings, *float* may be a scaling factor
- *2D-array* – The values determine the linewidth for each plotted arrow. Note that the shape has to match the one of u and v.

See also:

`arrowsize`, `arrowstyle`, `density`, `color`

`name = 'Linewidth of the arrows'`

`plot`
plot Formatoption instance in the plotter

`transpose`
transpose Formatoption instance in the plotter

`update`(*value*)
Method that is call to update the formatoption on the axes

Parameters **`value`** – Value to update

class `psy_simple.plotters.`**`VectorPlot`**(**args*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Choose the vector plot type

Possible types

Methods

add2format_coord(x, y)	Additional information for the `format_coord()`
get_xyz_1d(xcoord, x, ycoord, y, u, v)	Get closest x, y and z for the given *x* and *y* in *data* for
get_xyz_2d(xcoord, x, ycoord, y, u, v)	Get closest x, y and z for the given *x* and *y* in *data* for
get_xyz_tri(xcoord, x, ycoord, y, u, v)	Get closest x, y and z for the given *x* and *y* in *data* for
make_plot()	
remove()	Method to remove the effects of this formatoption
update(value)	Method that is call to update the formatoption on the axes

Attributes

arrowsize	arrowsize Formatoption instance in the plotter
arrowstyle	arrowstyle Formatoption instance in the plotter
bounds	bounds Formatoption instance in the plotter
children	list() -> new empty list
cmap	cmap Formatoption instance in the plotter
color	color Formatoption instance in the plotter

Continued on next page

Table 287 – continued from previous page

connections	list() -> new empty list
data_dependent	bool(x) -> bool
density	density Formatoption instance in the plotter
format_coord	The function that can replace the axes.format_coord method
group	str(object='') -> str
linewidth	linewidth Formatoption instance in the plotter
mappable	The mappable, i.e.
name	str(object='') -> str
plot_fmt	bool(x) -> bool
priority	int(x=0) -> integer
transform	transform Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter
xcoord	The x coordinate `xarray.Variable`
ycoord	The y coordinate `xarray.Variable`

str – Plot types can be either

quiver to make a quiver plot

stream to make a stream plot

add2format_coord (*x*, *y*)
Additional information for the `format_coord()`

arrowsize
arrowsize Formatoption instance in the plotter

arrowstyle
arrowstyle Formatoption instance in the plotter

bounds
bounds Formatoption instance in the plotter

children = ['cmap', 'bounds']

cmap
cmap Formatoption instance in the plotter

color
color Formatoption instance in the plotter

connections = ['transpose', 'transform', 'arrowsize', 'arrowstyle', 'density', 'linewi

data_dependent = True

density
density Formatoption instance in the plotter

format_coord
The function that can replace the axes.format_coord method

get_xyz_1d (*xcoord*, *x*, *ycoord*, *y*, *u*, *v*)
Get closest x, y and z for the given *x* and *y* in *data* for 1d coords

get_xyz_2d (*xcoord*, *x*, *ycoord*, *y*, *u*, *v*)
Get closest x, y and z for the given *x* and *y* in *data* for 2d coords

get_xyz_tri (*xcoord*, *x*, *ycoord*, *y*, *u*, *v*)
Get closest x, y and z for the given *x* and *y* in *data* for 1d coords

group = 'plotting'

linewidth
linewidth Formatoption instance in the plotter

make_plot()

mappable
The mappable, i.e. the container of the plot

name = 'Plot type of the arrows'

plot_fmt = True

priority = 20

remove()
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

transform
transform Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

xcoord
The x coordinate `xarray.Variable`

ycoord
The y coordinate `xarray.Variable`

class psy_simple.plotters.**ViolinPlot**(**args*, ***kwargs*)
Bases: `psyplot.plotter.Formatoption`

Choose how to make the violin plot

Possible types

Attributes

children	list() -> new empty list
color	color Formatoption instance in the plotter
data_dependent	bool(x) -> bool
group	str(object='') -> str
name	str(object='') -> str
plot_fmt	bool(x) -> bool
priority	int(x=0) -> integer
transpose	transpose Formatoption instance in the plotter

Methods

make_plot()	
remove()	Method to remove the effects of this formatoption
update(value)	Method that is call to update the formatoption on the axes

- *None or False* – Don't make any plotting
- *bool* – If True, visualize the violins

children = ['color', 'transpose']

color
color Formatoption instance in the plotter

data_dependent = True

group = 'plotting'

make_plot()

name = 'Violin plot type'

plot_fmt = True

priority = 20

remove()
Method to remove the effects of this formatoption

This method is called when the axes is cleared due to a formatoption with `requires_clearing` set to True. You don't necessarily have to implement this formatoption if your plot results are removed by the usual `matplotlib.axes.Axes.clear()` method.

transpose
transpose Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**ViolinPlotter**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: *psy_simple.plotters.SimplePlotterBase*

Plotter for making violin plots

Parameters

- **data** (*InteractiveArray or ArrayList, optional*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **ax** (*matplotlib.axes.Axes*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **auto_update** (*bool*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.

- **draw** (*bool or None*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **make_plot** (*bool*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **clear** (*bool*) – If True, the axes is cleared first
- **enable_post** (*bool*) – If True, the `post` formatoption is enabled and post processing scripts are allowed
- ****kwargs** – Any formatoption key from the `formatoptions` attribute that shall be used

Plot formatoptions

`plot`	Choose how to make the violin plot

Axes formatoptions

`xlim`	Set the x-axis limits
`ylim`	Set the y-axis limits
`axiscolor`	Color the x- and y-axes
`grid`	Display the grid
`tight`	Automatically adjust the plots.
`transpose`	Switch x- and y-axes

Axis tick formatoptions

`xticklabels`	Modify the x-axis ticklabels
`xticks`	Modify the x-axis ticks
`yticklabels`	Modify the x-axis ticklabels
`yticks`	Modify the y-axis ticks
`xrotation`	Rotate the x-axis ticks
`xtickprops`	Specify the x-axis tick parameters
`yrotation`	Rotate the y-axis ticks
`ytickprops`	Specify the y-axis tick parameters

Color coding formatoptions

`color`	Set the color coding

Masking formatoptions

`maskbetween`	Mask data points between two numbers
`maskgeq`	Mask data points greater than or equal to a number
`maskgreater`	Mask data points greater than a number
`maskleq`	Mask data points smaller than or equal to a number
`maskless`	Mask data points smaller than a number

Label formatoptions

figtitle	Plot a figure title
figtitleprops	Properties of the figure title
figtitlesize	Set the size of the figure title
figtitleweight	Set the fontweight of the figure title
labelprops	Set the font properties of both, x- and y-label
labelsize	Set the size of both, x- and y-label
labelweight	Set the font size of both, x- and y-label
text	Add text anywhere on the plot
title	Show the title
titleprops	Properties of the title
titlesize	Set the size of the title
titleweight	Set the fontweight of the title
xlabel	Set the x-axis label
ylabel	Set the y-axis label

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

Miscallaneous formatoptions

legend	Draw a legend
legendlabels	Set the labels of the arrays in the legend
sym_lims	Make x- and y-axis symmetric
ticksize	Change the ticksize of the ticklabels
tickweight	Change the fontweight of the ticks

plot
Choose how to make the violin plot

Possible types

- *None or False* – Don't make any plotting
- *bool* – If True, visualize the violins

xlim
Set the x-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum.

The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

minmax Uses the minimum and maximum

sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

`xticklabels`
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

`xticks`
Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

hour draw ticks every hour

day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

xticklabels, *ticksize*, *tickweight*, *xtickprops*, *yticks*

ylim
Set the y-axis limits

Possible types

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

minmax Uses the minimum and maximum

sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`xlim`

`yticklabels`
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

`yticks`
Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, …]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

hour draw ticks every hour

day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

`yticklabels`, *`ticksize`*, *`tickweight`*, *`ytickprops`*

`xticks` for possible examples

`color`

Set the color coding

This formatoptions sets the color of the lines, bars, etc.

Possible types

- *None* – to use the axes color_cycle
- *iterable* – (e.g. list) to specify the colors manually
- *str* – Strings may be any valid colormap name suitable for the `matplotlib.cm.get_cmap()` function or one of the color lists defined in the 'colors.cmaps' key of the `psyplot.rcParams` dictionary (including their reversed color maps given via the '_r' extension).
- *matplotlib.colors.ColorMap* – to automatically choose the colors according to the number of lines, etc. from the given colormap

`maskbetween`

Mask data points between two numbers

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskgeq*

maskgeq
Mask data points greater than or equal to a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgreater*, *maskbetween*

maskgreater
Mask data points greater than a number

Possible types

float – The floating number to mask above

See also:

maskless, *maskleq*, *maskgeq*, *maskbetween*

maskleq
Mask data points smaller than or equal to a number

Possible types

float – The floating number to mask below

See also:

maskless, *maskgreater*, *maskgeq*, *maskbetween*

maskless
Mask data points smaller than a number

Possible types

float – The floating number to mask below

See also:

maskleq, *maskgreater*, *maskgeq*, *maskbetween*

figtitle
Plot a figure title

Set the title of the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)

- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `suptitle()` function

Notes

- If the plotter is part of a `psyplot.project.Project` and multiple plotters of this project are on the same figure, the replacement attributes (see above) are joined by a delimiter. If the `delimiter` attribute of this `Figtitle` instance is not None, it will be used. Otherwise the rcParams['texts.delimiter'] item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *`title`* formatoption.

See also:

`title`, *`figtitlesize`*, *`figtitleweight`*, *`figtitleprops`*

`figtitleprops`
Properties of the figure title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`figtitle`, *`figtitlesize`*, *`figtitleweight`*

`figtitlesize`
Set the size of the figure title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`figtitle`, *`figtitleweight`*, *`figtitleprops`*

figtitleweight

Set the fontweight of the figure title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`figtitle`, `figtitlesize`, `figtitleprops`

labelprops

Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

`xlabel`, `ylabel`, `labelsize`, `labelweight`

labelsize

Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`xlabel`, `ylabel`, `labelweight`, `labelprops`

labelweight

Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

xlabel, *ylabel*, *labelsize*, *labelprops*

text

Add text anywhere on the plot

This formatoption draws a text on the specified position on the figure. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – If string s: this will be used as (1., 1., s, {'ha': 'right'}) (i.e. a string in the upper right corner of the axes).
- *tuple or list of tuples (x,y,s[,coord.-system][,options]])* – Each tuple defines a text instance on the plot. 0<=x, y<=1 are the coordinates. The coord.-system can be either the data coordinates (default, `'data'`) or the axes coordinates (`'axes'`) or the figure coordinates ('fig'). The string s finally is the text. options may be a dictionary to specify format the appearence (e.g. `'color'`, `'fontweight'`, `'fontsize'`, etc., see `matplotlib.text.Text` for possible keys). To remove one single text from the plot, set (x,y,''[, coord.-system]) for the text at position (x,y)
- *empty list* – remove all texts from the plot

See also:

title, *figtitle*

title

Show the title

Set the title of the plot. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The title for the `title()` function.

Notes

This is the title of this specific subplot! For the title of the whole figure, see the *`figtitle`* formatoption.

See also:

`figtitle`, *`titlesize`*, *`titleweight`*, *`titleprops`*

`titleprops`
Properties of the title

Specify the font properties of the figure title manually.

Possible types

dict – Items may be any valid text property

See also:

`title`, *`titlesize`*, *`titleweight`*

`titlesize`
Set the size of the title

Possible types

- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

`title`, *`titleweight`*, *`titleprops`*

titleweight
Set the fontweight of the title

Possible types

- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

title, *titlesize*, *titleprops*

xlabel
Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

ylabel
Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)

- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

post

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

axiscolor

Color the x- and y-axes

This formatoption colors the left, right, bottom and top axis bar.

Possible types

dict – Keys may be one of {'right', 'left', 'bottom', 'top'}, the values can be any valid color or None.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

grid
Display the grid

Show the grid on the plot with the specified color.

Possible types

- *None* – If the grid is currently shown, it will not be displayed any longer. If the grid is not shown, it will be drawn
- *bool* – If True, the grid is displayed with the automatic settings (usually black)
- *string, tuple.* – Defines the color of the grid.

Notes

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

In addition, you can specify colors in many weird and wonderful ways, including full names (`'green'`), hex strings (`'#008000'`), RGB or RGBA tuples (`(0,1,0,1)`) or grayscale intensities as a string (`'0.8'`).

tight
Automatically adjust the plots.

If set to True, the plots are automatically adjusted to fit to the figure limitations via the `matplotlib.pyplot.tight_layout()` function.

Possible types

bool – True for automatic adjustment

> **Warning:** There is no update method to undo what happend after this formatoption is set to True!

transpose
Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

xrotation
Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

yrotation

xtickprops
Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *ytickprops*

yrotation
Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

xrotation

ytickprops
Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *xtickprops*

legend
Draw a legend

This formatoption determines where and if to draw the legend. It uses the `labels` formatoption to determine the labels.

Possible types

- *bool* – Draw a legend or not
- *str or int* – Specifies where to plot the legend (i.e. the location)
- *dict* – Give the keywords for the `matplotlib.pyplot.legend()` function

See also:

`labels`

legendlabels
Set the labels of the arrays in the legend

This formatoption specifies the labels for each array in the legend. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams 'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

- *str* – A single string that shall be used for all arrays.
- *list of str* – Same as a single string but specified for each array

See also:

legend

sym_lims
Make x- and y-axis symmetric

Possible types

- *None* – No symmetric type
- *'min'* – Use the minimum of x- and y-limits
- *'max'* – Use the maximum of x- and y-limits
- *[str, str]* – A combination, `None`, `'min'` and `'max'` specific for minimum and maximum limit

ticksize
Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

tickweight, *xtickprops*, *ytickprops*

tickweight
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

`ticksize`, `xtickprops`, `ytickprops`

class psy_simple.plotters.**ViolinXTickLabels**(**args*, ***kwargs*)

Bases: `psy_simple.plotters.XTickLabels`, `psy_simple.base.TextBase`

Modify the x-axis ticklabels

Possible types

Attributes

`data_dependent`	bool(x) -> bool
`transpose`	transpose Formatoption instance in the plotter
`xticks`	xticks Formatoption instance in the plotter
`yticklabels`	yticklabels Formatoption instance in the plotter

Methods

`update_axis`(value)

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

data_dependent = True

transpose

transpose Formatoption instance in the plotter

update_axis(*value*)

xticks

xticks Formatoption instance in the plotter

yticklabels

yticklabels Formatoption instance in the plotter

class psy_simple.plotters.**ViolinXTicks**(**args*, ***kwargs*)

Bases: `psy_simple.plotters.XTicks`

Modify the x-axis ticks

Possible types

Attributes

`array`	The numpy array of the data
`plot`	plot Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter
`yticks`	yticks Formatoption instance in the plotter

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

`xticklabels`, `ticksize`, `tickweight`, `xtickprops`, *`yticks`*

`array`
The numpy array of the data

`plot`
plot Formatoption instance in the plotter

`transpose`
transpose Formatoption instance in the plotter

`yticks`
yticks Formatoption instance in the plotter

class `psy_simple.plotters.`**`ViolinXlim`**`(*args, **kwargs)`
Bases: *`psy_simple.plotters.Xlim`*

Set the x-axis limits

Possible types

Attributes

`array`	The numpy array of the data
`plot`	plot Formatoption instance in the plotter
`sym_lims`	sym_lims Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter
`xticks`	xticks Formatoption instance in the plotter
`ylim`	ylim Formatoption instance in the plotter

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

array
The numpy array of the data

plot
plot Formatoption instance in the plotter

sym_lims
sym_lims Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

xticks
xticks Formatoption instance in the plotter

ylim
ylim Formatoption instance in the plotter

class psy_simple.plotters.**ViolinYTickLabels**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.YTickLabels*, *psy_simple.base.TextBase*

Modify the x-axis ticklabels

Possible types

Attributes

data_dependent	bool(x) -> bool
transpose	transpose Formatoption instance in the plotter
yticks	yticks Formatoption instance in the plotter

Methods

update_axis(value)	

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

data_dependent = True

transpose
transpose Formatoption instance in the plotter

update_axis(*value*)

yticks
yticks Formatoption instance in the plotter

class psy_simple.plotters.**ViolinYTicks**(**args*, ***kwargs*)

Bases: `psy_simple.plotters.YTicks`

Modify the y-axis ticks

Possible types

Attributes

`array`	The numpy array of the data
`plot`	plot Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

`yticklabels`, `ticksize`, `tickweight`, `ytickprops`

`xticks` for possible examples

array
The numpy array of the data

plot
plot Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

class psy_simple.plotters.**ViolinYlim**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.Ylim*

Set the y-axis limits

Possible types

Attributes

array	The numpy array of the data
plot	plot Formatoption instance in the plotter
sym_lims	sym_lims Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter
xlim	xlim Formatoption instance in the plotter
yticks	yticks Formatoption instance in the plotter

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

xlim

array
The numpy array of the data

plot
plot Formatoption instance in the plotter

sym_lims
sym_lims Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

xlim
xlim Formatoption instance in the plotter

yticks
yticks Formatoption instance in the plotter

class psy_simple.plotters.**XRotation**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: psyplot.plotter.Formatoption

Rotate the x-axis ticks

Possible types

Attributes

children	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
yticklabels	yticklabels Formatoption instance in the plotter

Methods

update(value)	Method that is call to update the formatoption on the axes

float – The rotation angle in degrees

See also:

yrotation

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the *children* attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the dependencies attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the *children*, dependencies and connections attributes, with values being the name of the new formatoption in this plotter.

children = ['yticklabels']

group = 'ticks'

name = 'Rotate x-ticklabels'

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

yticklabels
yticklabels Formatoption instance in the plotter

class psy_simple.plotters.**XTickLabels**(**args*, ***kwargs*)
Bases: `psy_simple.plotters.TickLabels`

Modify the x-axis ticklabels

Possible types

Attributes

`axis`	The axis on the axes to modify the ticks of
`dependencies`	list() -> new empty list
`name`	str(object='') -> str
`transpose`	transpose Formatoption instance in the plotter
`xticks`	xticks Formatoption instance in the plotter
`yticklabels`	yticklabels Formatoption instance in the plotter

Methods

`initialize_plot`(*args, **kwargs)	Method that is called when the plot is made the first time

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`xticks`, `ticksize`, `tickweight`, `xtickprops`, `yticklabels`

axis
The axis on the axes to modify the ticks of

dependencies = ['transpose', 'xticks', 'yticklabels']

initialize_plot(**args*, ***kwargs*)
Method that is called when the plot is made the first time

Parameters **value** – The value to use for the initialization

name = 'x-xxis Ticklabels'

transpose
transpose Formatoption instance in the plotter

xticks
xticks Formatoption instance in the plotter

yticklabels
yticklabels Formatoption instance in the plotter

class psy_simple.plotters.**XTickProps**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.plotters.TickPropsBase*, *psy_simple.plotters.TicksManager*, psyplot.plotter.DictFormatoption

Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

Attributes

axis	
axisname	str(object='') -> str
name	str(object='') -> str

- *dict* – A dictionary with the keys 'minor' and (or) 'major' to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams 'ticks.which' key (usually 'major'). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the matplotlib.pyplot.tick_params() function

See also:

xticks, yticks, ticksize, tickweight, ytickprops

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a psyplot.InteractiveList
- **additional_children** (*list or str*) – Additional children to use (see the children attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the dependencies attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the children, dependencies and connections attributes, with values being the name of the new formatoption in this plotter.

axis

axisname = 'x'

name = 'Font properties of the x-ticklabels'

class psy_simple.plotters.**XTicks**(**args*, ***kwargs*)

Bases: *psy_simple.plotters.DtTicksBase*

Modify the x-axis ticks

Possible types

Attributes

axis	
children	list() -> new empty list
data	The data that is plotted
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter
yticks	yticks Formatoption instance in the plotter

Methods

initialize_plot(*args, **kwargs)	Method that is called when the plot is made the first time

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

`xticklabels`, `ticksize`, `tickweight`, `xtickprops`, *`yticks`*

`axis`

`children = ['yticks']`

`data`
The data that is plotted

`initialize_plot`(**args*, ***kwargs*)
Method that is called when the plot is made the first time

Parameters **`value`** – The value to use for the initialization

`name = 'Location of the x-Axis ticks'`

`plot`
plot Formatoption instance in the plotter

`transpose`
transpose Formatoption instance in the plotter

`yticks`
yticks Formatoption instance in the plotter

class `psy_simple.plotters.`**`XTicks2D`**(**args*, ***kwargs*)
Bases: *`psy_simple.plotters.XTicks`*

Modify the x-axis ticks

Possible types

Attributes

`data`	The data that is plotted
`plot`	plot Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter
`yticks`	yticks Formatoption instance in the plotter

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

`xticklabels`, `ticksize`, `tickweight`, `xtickprops`, *`yticks`*

`data`
The data that is plotted

`plot`
plot Formatoption instance in the plotter

`transpose`
transpose Formatoption instance in the plotter

`yticks`
yticks Formatoption instance in the plotter

class `psy_simple.plotters.`**`XYTickPlotter`**(*data=None*, *ax=None*, *auto_update=None*, *project=None*, *draw=None*, *make_plot=True*, *clear=False*, *enable_post=False*, ***kwargs*)

Bases: `psyplot.plotter.Plotter`

Plotter class for x- and y-ticks and x- and y- ticklabels

Parameters

- **`data`** (*`InteractiveArray or ArrayList, optional`*) – Data object that shall be visualized. If given and *plot* is True, the `initialize_plot()` method is called at the end. Otherwise you can call this method later by yourself
- **`ax`** (*`matplotlib.axes.Axes`*) – Matplotlib Axes to plot on. If None, a new one will be created as soon as the `initialize_plot()` method is called
- **`auto_update`** (*`bool`*) – Default: None. A boolean indicating whether this list shall automatically update the contained arrays when calling the `update()` method or not. See also the `no_auto_update` attribute. If None, the value from the `'lists.auto_update'` key in the `psyplot.rcParams` dictionary is used.
- **`draw`** (*`bool or None`*) – Boolean to control whether the figure of this array shall be drawn at the end. If None, it defaults to the *'auto_draw'*' parameter in the `psyplot.rcParams` dictionary
- **`make_plot`** (*`bool`*) – If True, and *data* is not None, the plot is initialized. Otherwise only the framework between plotter and data is set up
- **`clear`** (*`bool`*) – If True, the axes is cleared first
- **`enable_post`** (*`bool`*) – If True, the *`post`* formatoption is enabled and post processing scripts are allowed
- **`**kwargs`** – Any formatoption key from the `formatoptions` attribute that shall be used

Label formatoptions

`labelprops`	Set the font properties of both, x- and y-label

Continued on next page

Table 314 – continued from previous page

labelsize	Set the size of both, x- and y-label
labelweight	Set the font size of both, x- and y-label
xlabel	Set the x-axis label
ylabel	Set the y-axis label

Miscallaneous formatoptions

ticksize	Change the ticksize of the ticklabels
tickweight	Change the fontweight of the ticks

Axes formatoptions

transpose	Switch x- and y-axes

Axis tick formatoptions

xrotation	Rotate the x-axis ticks
xticklabels	Modify the x-axis ticklabels
xtickprops	Specify the x-axis tick parameters
xticks	Modify the x-axis ticks
yrotation	Rotate the y-axis ticks
yticklabels	Modify the y-axis ticklabels
ytickprops	Specify the y-axis tick parameters
yticks	Modify the y-axis ticks

Post processing formatoptions

post	Apply your own postprocessing script
post_timing	Determine when to run the `post` formatoption

labelprops
Set the font properties of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *dict* – Items may be any valid text property

See also:

xlabel, *ylabel*, *labelsize*, *labelweight*

labelsize
Set the size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

xlabel, *ylabel*, *labelweight*, *labelprops*

labelweight
Set the font size of both, x- and y-label

Possible types

- *dict* – A dictionary with the keys `'x'` and (or) `'y'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is used for the x- and y-axis. The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

xlabel, *ylabel*, *labelsize*, *labelprops*

ticksize
Change the ticksize of the ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – The absolute font size in points (e.g., 12)
- *string* – Strings might be 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'.

See also:

tickweight, *xtickprops*, *ytickprops*

tickweight
Change the fontweight of the ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *float* – a float between 0 and 1000
- *string* – Possible strings are one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

See also:

ticksize, *xtickprops*, *ytickprops*

transpose
Switch x- and y-axes

By default, one-dimensional arrays have the dimension on the x-axis and two dimensional arrays have the first dimension on the y and the second on the x-axis. You can set this formatoption to True to change this behaviour

Possible types

bool – If True, axes are switched

xlabel
Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

xrotation
Rotate the x-axis ticks

Possible types

float – The rotation angle in degrees

See also:

yrotation

xticklabels
Modify the x-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

xticks, *ticksize*, *tickweight*, *xtickprops*, *yticklabels*

xtickprops
Specify the x-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters on the x-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *ytickprops*

xticks
Modify the x-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

Examples

Plot 11 ticks over the whole data range:

```
>>> plotter.update(xticks='rounded')
```

Plot 7 ticks over the whole data range where the maximal and minimal tick matches the data maximum and minimum:

```
>>> plotter.update(xticks=['minmax', 7])
```

Plot ticks every year and minor ticks every month:

```
>>> plotter.update(xticks={'major': 'year', 'minor': 'month'})
```

See also:

xticklabels, *ticksize*, *tickweight*, *xtickprops*, *yticks*

ylabel

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

yrotation

Rotate the y-axis ticks

Possible types

float – The rotation angle in degrees

See also:

xrotation

yticklabels

Modify the y-axis ticklabels

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

yticks, *ticksize*, *tickweight*, *ytickprops*, *xticklabels*

ytickprops

Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

xticks, *yticks*, *ticksize*, *tickweight*, *xtickprops*

yticks

Modify the y-axis ticks

Possible types

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, . . .]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

minmax Uses the minimum as minimal tick and maximum as maximal tick

sym Same as minmax but symmetric around zero

hour draw ticks every hour

day draw ticks every day

week draw ticks every week

month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

`yticklabels`, `ticksize`, `tickweight`, `ytickprops`

`xticks` for possible examples

`post`

Apply your own postprocessing script

This formatoption let's you apply your own post processing script. Just enter the script as a string and it will be executed. The formatoption will be made available via the `self` variable

Possible types

- *None* – Don't do anything
- *str* – The post processing script as string

Note: This formatoption uses the built-in `exec()` function to compile the script. Since this poses a security risk when loading psyplot projects, it is by default disabled through the `Plotter.enable_post` attribute. If you are sure that you can trust the script in this formatoption, set this attribute of the corresponding `Plotter` to `True`

Examples

Assume, you want to manually add the mean of the data to the title of the matplotlib axes. You can simply do this via

```
from psyplot.plotter import Plotter
from xarray import DataArray
plotter = Plotter(DataArray([1, 2, 3]))
# enable the post formatoption
plotter.enable_post = True
plotter.update(post="self.ax.set_title(str(self.data.mean()))")
plotter.ax.get_title()
'2.0'
```

By default, the `post` formatoption is only ran, when it is explicitly updated. However, you can use the *post_timing* formatoption, to run it automatically. E.g. for running it after every update of the plotter, you can set

```
plotter.update(post_timing='always')
```

See also:

post_timing Determine the timing of this formatoption

post_timing

Determine when to run the *post* formatoption

This formatoption determines, whether the *post* formatoption should be run never, after replot or after every update.

Possible types

- *'never'* – Never run post processing scripts
- *'always'* – Always run post processing scripts
- *'replot'* – Only run post processing scripts when the data changes or a replot is necessary

See also:

post The post processing formatoption

class psy_simple.plotters.**Xlabel**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.base.TextBase*, psyplot.plotter.Formatoption

Set the x-axis label

Set the label for the x-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are

- dtinfo: `%B %d, %Y. %H:%M`
- desc: `%(long_name)s [%(units)s]`
- dinfo: `%B %d, %Y`
- tinfo: `%H:%M`
- sdesc: `%(name)s [%(units)s]`

Attributes

children	list() -> new empty list
enhanced_attrs	The enhanced attributes of the array
name	str(object='') -> str
transpose	transpose Formatoption instance in the plotter
ylabel	ylabel Formatoption instance in the plotter

Methods

initialize_plot(value)	Method that is called when the plot is made the first time
update(value)	Method that is call to update the formatoption on the axes

Possible types

str – The text for the `xlabel()` function.

See also:

`xlabelsize`, `xlabelweight`, `xlabelprops`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the *children* attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the *children*, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

children = ['transpose', 'ylabel']

enhanced_attrs

The enhanced attributes of the array

initialize_plot(*value*)
Method that is called when the plot is made the first time

Parameters **value** – The value to use for the initialization

name = 'x-axis label'

transpose
transpose Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

ylabel
ylabel Formatoption instance in the plotter

class psy_simple.plotters.**Xlim**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.LimitBase*

Set the x-axis limits

Possible types

Attributes

array	The numpy array of the data
axisname	str(object='') -> str
children	list() -> new empty list
connections	list() -> new empty list
dependencies	list() -> new empty list
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
sym_lims	sym_lims Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter
xticks	xticks Formatoption instance in the plotter
ylim	ylim Formatoption instance in the plotter

Methods

initialize_plot(value)	Method that is called when the plot is made the first time
set_limit(*args)	The method to set the minimum and maximum limit

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

minmax Uses the minimum and maximum

sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

array
The numpy array of the data

axisname = 'x'

children = ['transpose', 'ylim']

connections = ['plot', 'sym_lims']

dependencies = ['xticks']

initialize_plot (*value*)
Method that is called when the plot is made the first time

Parameters value – The value to use for the initialization

name = 'x-axis limits'

plot
plot Formatoption instance in the plotter

set_limit (**args*)
The method to set the minimum and maximum limit

Parameters

- **min_val** (`float`) – The value for the lower limit
- **max_val** (`float`) – The value for the upper limit

sym_lims
sym_lims Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

xticks
xticks Formatoption instance in the plotter

ylim
ylim Formatoption instance in the plotter

class psy_simple.plotters.**Xlim2D** (**args*, ***kwargs*)
Bases: `psy_simple.plotters.Xlim`

Set the x-axis limits

Possible types

Attributes

`array`	The numpy array of the data
`plot`	plot Formatoption instance in the plotter
`sym_lims`	sym_lims Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter
`xticks`	xticks Formatoption instance in the plotter
`ylim`	ylim Formatoption instance in the plotter

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero
- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`ylim`

`array`
The numpy array of the data

`plot`
plot Formatoption instance in the plotter

`sym_lims`
sym_lims Formatoption instance in the plotter

`transpose`
transpose Formatoption instance in the plotter

`xticks`
xticks Formatoption instance in the plotter

`ylim`
ylim Formatoption instance in the plotter

class `psy_simple.plotters.`**`YRotation`**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psyplot.plotter.Formatoption`

Rotate the y-axis ticks

Possible types

Attributes

children	list() -> new empty list
group	str(object='') -> str
name	str(object='') -> str
yticklabels	yticklabels Formatoption instance in the plotter

Methods

update(value)	Method that is call to update the formatoption on the axes

float – The rotation angle in degrees

See also:

`xrotation`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the *children* attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the *children*, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

children = ['yticklabels']

group = 'ticks'

name = 'Rotate y-ticklabels'

update(*value*)
Method that is call to update the formatoption on the axes

Parameters **value** – Value to update

yticklabels
yticklabels Formatoption instance in the plotter

class psy_simple.plotters.**YTickLabels**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.TickLabels*

Modify the y-axis ticklabels

Possible types

Attributes

`axis`	The axis on the axes to modify the ticks of
`dependencies`	list() -> new empty list
`name`	str(object='') -> str
`transpose`	transpose Formatoption instance in the plotter
`yticks`	yticks Formatoption instance in the plotter

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *str* – A formatstring like `'%Y'` for plotting the year (in the case that time is shown on the axis) or '%i' for integers
- *array* – An array of strings to use for the ticklabels

See also:

`yticks`, `ticksize`, `tickweight`, `ytickprops`, `xticklabels`

axis
The axis on the axes to modify the ticks of

dependencies = ['transpose', 'yticks']

name = 'y-xxis ticklabels'

transpose
transpose Formatoption instance in the plotter

yticks
yticks Formatoption instance in the plotter

class psy_simple.plotters.**YTickProps**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: `psy_simple.plotters.XTickProps`

Specify the y-axis tick parameters

This formatoption can be used to make a detailed change of the ticks parameters of the y-axis.

Possible types

Attributes

`axis`	
`axisname`	str(object='') -> str
`name`	str(object='') -> str

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.

- *dict* – Items may be anything of the `matplotlib.pyplot.tick_params()` function

See also:

`xticks`, `yticks`, `ticksize`, `tickweight`, `xtickprops`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.
- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

axis

axisname = 'y'

name = 'Font properties of the y-ticklabels'

class psy_simple.plotters.**YTicks**(**args*, ***kwargs*)

Bases: *psy_simple.plotters.DtTicksBase*

Modify the y-axis ticks

Possible types

Attributes

axis	
data	The data that is plotted
name	str(object='') -> str
plot	plot Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used

- *numeric array* – specifies the ticks manually
- *str or list [str, ...]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

`yticklabels`, `ticksize`, `tickweight`, `ytickprops`

`xticks` for possible examples

`axis`

`data`
The data that is plotted

`name = 'Location of the y-Axis ticks'`

`plot`
plot Formatoption instance in the plotter

`transpose`
transpose Formatoption instance in the plotter

class `psy_simple.plotters.`**`YTicks2D`**(**args*, ***kwargs*)
Bases: *`psy_simple.plotters.YTicks`*

Modify the y-axis ticks

Possible types

Attributes

`data`	The data that is plotted
`plot`	plot Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter

- *dict* – A dictionary with the keys `'minor'` and (or) `'major'` to specify which ticks are managed. If the given value is not a dictionary with those keys, it is put into a dictionary with the key determined by the rcParams `'ticks.which'` key (usually `'major'`). The values in the dictionary can be one types below.
- *None* – use the default ticks
- *int* – for an integer *i*, only every *i-th* tick of the default ticks are used
- *numeric array* – specifies the ticks manually
- *str or list [str, …]* – Automatically determine the ticks corresponding to the data. The given string determines how the ticks are calculated. If not a single string but a list, the second value determines the number of ticks (see below). A string can be one of the following:

  data plot the ticks exactly where the data is.

  mid plot the ticks in the middle of the data.

  rounded Sets the minimum and maximum of the ticks to the rounded data minimum or maximum. Ticks are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimal tick will always be lower or equal than the data minimum, the maximal tick will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the ticks are chose such that they are symmetric around zero

  minmax Uses the minimum as minimal tick and maximum as maximal tick

  sym Same as minmax but symmetric around zero

  hour draw ticks every hour

  day draw ticks every day

  week draw ticks every week

  month, monthend, monthbegin draw ticks in the middle, at the end or at the beginning of each month

  year, yearend, yearbegin draw ticks in the middle, at the end or at the beginning of each year

  For data, mid, hour, day, week, month, etc., the optional second value can be an integer i determining that every i-th data point shall be used (by default, it is set to 1). For rounded, roundedsym, minmax and sym, the second value determines the total number of ticks (defaults to 11).

See also:

`yticklabels`, `ticksize`, `tickweight`, `ytickprops`

`xticks` for possible examples

`data`
The data that is plotted

`plot`
plot Formatoption instance in the plotter

`transpose`
transpose Formatoption instance in the plotter

class psy_simple.plotters.**Ylabel**(*key*, *plotter=None*, *index_in_list=None*, *additional_children=[]*, *additional_dependencies=[]*, ***kwargs*)

Bases: *psy_simple.base.TextBase*, psyplot.plotter.Formatoption

Set the y-axis label

Set the label for the y-axis. You can insert any meta key from the `xarray.DataArray.attrs` via a string like `'%(key)s'`. Furthermore there are some special cases:

- Strings like `'%Y'`, `'%b'`, etc. will be replaced using the `datetime.datetime.strftime()` method as long as the data has a time coordinate and this can be converted to a `datetime` object.
- `'%(x)s'`, `'%(y)s'`, `'%(z)s'`, `'%(t)s'` will be replaced by the value of the x-, y-, z- or time coordinate (as long as this coordinate is one-dimensional in the data)
- any attribute of one of the above coordinates is inserted via `axis + key` (e.g. the name of the x-coordinate can be inserted via `'%(xname)s'`).
- Labels defined in the `psyplot.rcParams` `'texts.labels'` key are also replaced when enclosed by '{}'. The standard labels are
  - dtinfo: `%B %d, %Y. %H:%M`
  - desc: `%(long_name)s [%(units)s]`
  - dinfo: `%B %d, %Y`
  - tinfo: `%H:%M`
  - sdesc: `%(name)s [%(units)s]`

Attributes

children	list() -> new empty list
enhanced_attrs	The enhanced attributes of the array
name	str(object='') -> str
transpose	transpose Formatoption instance in the plotter

Methods

initialize_plot(value)	Method that is called when the plot is made the first time
update(value)	Method that is call to update the formatoption on the axes

Possible types

str – The text for the `ylabel()` function.

See also:

`ylabelsize`, `ylabelweight`, `ylabelprops`

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psyplot.plotter.Plotter*) – Plotter instance that holds this formatoption. If None, it is assumed that this instance serves as a descriptor.

- **index_in_list** (*int or None*) – The index that shall be used if the data is a `psyplot.InteractiveList`
- **additional_children** (*list or str*) – Additional children to use (see the `children` attribute)
- **additional_dependencies** (*list or str*) – Additional dependencies to use (see the `dependencies` attribute)
- ****kwargs** – Further keywords may be used to specify different names for children, dependencies and connection formatoptions that match the setup of the plotter. Hence, keywords may be anything of the `children`, `dependencies` and `connections` attributes, with values being the name of the new formatoption in this plotter.

children = ['transpose']

enhanced_attrs
The enhanced attributes of the array

initialize_plot(*value*)
Method that is called when the plot is made the first time

Parameters value – The value to use for the initialization

name = 'y-axis label'

transpose
transpose Formatoption instance in the plotter

update(*value*)
Method that is call to update the formatoption on the axes

Parameters value – Value to update

class psy_simple.plotters.**Ylim**(**args*, ***kwargs*)
Bases: `psy_simple.plotters.LimitBase`

Set the y-axis limits

Possible types

Attributes

`array`	The numpy array of the data
`axisname`	str(object=’‘) -> str
`children`	list() -> new empty list
`connections`	list() -> new empty list
`dependencies`	list() -> new empty list
`name`	str(object=’‘) -> str
`plot`	plot Formatoption instance in the plotter
`sym_lims`	sym_lims Formatoption instance in the plotter
`transpose`	transpose Formatoption instance in the plotter
`xlim`	xlim Formatoption instance in the plotter
`yticks`	yticks Formatoption instance in the plotter

Methods

`set_limit`(*args)	The method to set the minimum and maximum limit

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

`xlim`

`array`
The numpy array of the data

`axisname = 'y'`

`children = ['transpose', 'xlim']`

`connections = ['plot', 'sym_lims']`

`dependencies = ['yticks']`

`name = 'y-axis limits'`

`plot`
plot Formatoption instance in the plotter

`set_limit`(**args*)
The method to set the minimum and maximum limit

Parameters

- **`min_val`** (`float`) – The value for the lower limit
- **`max_val`** (`float`) – The value for the upper limit

`sym_lims`
sym_lims Formatoption instance in the plotter

`transpose`
transpose Formatoption instance in the plotter

`xlim`
xlim Formatoption instance in the plotter

`yticks`
yticks Formatoption instance in the plotter

class psy_simple.plotters.**Ylim2D**(**args*, ***kwargs*)
Bases: *psy_simple.plotters.Ylim*

Set the y-axis limits

Possible types

Attributes

array	The numpy array of the data
plot	plot Formatoption instance in the plotter
sym_lims	sym_lims Formatoption instance in the plotter
transpose	transpose Formatoption instance in the plotter
xlim	xlim Formatoption instance in the plotter
yticks	yticks Formatoption instance in the plotter

- *None* – To not change the current limits
- *str or list [str, str] or [[str, float], [str, float]]* – Automatically determine the ticks corresponding to the data. The given string determines how the limits are calculated. The float determines the percentile to use A string can be one of the following:

  rounded Sets the minimum and maximum of the limits to the rounded data minimum or maximum. Limits are rounded to the next 0.5 value with to the difference between data max- and minimum. The minimum will always be lower or equal than the data minimum, the maximum will always be higher or equal than the data maximum.

  roundedsym Same as *rounded* above but the limits are chosen such that they are symmetric around zero

  minmax Uses the minimum and maximum

  sym Same as minmax but symmetric around zero

- *tuple (xmin, xmax)* – *xmin* is the smaller value, *xmax* the larger. Any of those values can be None or one of the strings (or lists) above to use the corresponding value here

See also:

xlim

array
The numpy array of the data

plot
plot Formatoption instance in the plotter

sym_lims
sym_lims Formatoption instance in the plotter

transpose
transpose Formatoption instance in the plotter

xlim
xlim Formatoption instance in the plotter

yticks
yticks Formatoption instance in the plotter

`psy_simple.plotters.`**`format_coord_func`**(*ax*, *ref*)

Create a function that can replace the `matplotlib.axes.Axes.format_coord()`

Parameters

- **ax** (*matplotlib.axes.Axes*) – The axes instance
- **ref** (*weakref.weakref*) – The reference to the `Formatoption` instance

Returns The function that can be used to replace *ax.format_coord*

Return type function

`psy_simple.plotters.`**`round_to_05`**(*n*, *exp=None*, *mode='s'*)

Round to the next 0.5-value.

This function applies the round function *func* to round *n* to the next 0.5-value with respect to its exponent with base 10 (i.e. 1.3e-4 will be rounded to 1.5e-4) if *exp* is None or with respect to the given exponent in *exp*.

Parameters

- **n** (*numpy.ndarray*) – number to round
- **exp** (*int or numpy.ndarray*) – Exponent for rounding. If None, it will be computed from *n* to be the exponents for base 10.
- **mode** (*{'s', 'l'}*) – rounding mode. If 's', it will be rounded to value whose absolute value is below *n*, if 'l' it will rounded to the value whose absolute value is above *n*.

Returns rounded *n*

Return type numpy.ndarray

Examples

The effects of the different parameters are show in the example below:

```
>>> from psyplot.plotter.simple import round_to_05
>>> a = [-100.3, 40.6, 8.7, -0.00023]
>>>round_to_05(a, mode='s')
array([ -1.00000000e+02,   4.00000000e+01,   8.50000000e+00,
        -2.00000000e-04])

>>> round_to_05(a, mode='l')
array([ -1.50000000e+02,   4.50000000e+01,   9.00000000e+00,
        -2.50000000e-04])
```

---

psy_simple.plugin module

psy-simple psyplot plugin

This module defines the rcParams for the psy-simple plugin

Classes

BoundsValidator(*args, **kwargs)	For parameter description see `matplotlib.rcsetup.ValidateInStrings`.
DictValValidator(key, valid, validators, default)	A validation class for formatoptions that expect dictionaries as values

Continued on next page

Table 335 – continued from previous page

LineWidthValidator(key, valid[, ignorecase])	valid is a list of legal strings
TicksValidator(key, valid[, ignorecase])	valid is a list of legal strings
ValidateList([dtype, length, listtype])	Validate a list of the specified *dtype*

Functions

get_versions([requirements])	
patch_prior_1_0(plotter_d, versions)	Patch psy_simple plotters for versions smaller than 1.0
try_and_error(*funcs)	Apply multiple validation functions
validate_alpha(val)	Validate an alpha value between 0 and 1
validate_axiscolor(value)	Validate a dictionary containing axiscolor definitions
validate_cbarpos(value)	Validate a colorbar position
validate_cmap(val)	Validate a colormap
validate_cmaps(cmaps)	Validate a dictionary of color lists
validate_dataarray(val)	
validate_err_calc(val)	Validation function for the
validate_float(s)	convert *s* to float or raise
validate_fontweight(value)	
validate_iter(value)	Validate that the given value is an iterable
validate_legend(value)	
validate_limits(value)	
validate_lineplot(value)	Validate the value for the LinePlotter.plot formatoption
validate_marker(val)	Does not really make a validation because markers can be quite of
validate_none(b)	Validate that None is given
validate_str(s)	Validate a string
validate_sym_lims(val)	
validate_text(value)	Validate a text formatoption
validate_ticklabels(value)	

Data

patches	patches to apply when loading a project
rcParams	the `RcParams` for the psy-simple plugin

class psy_simple.plugin.**BoundsValidator**(**args*, ***kwargs*)

Bases: `matplotlib.rcsetup.ValidateInStrings`

For parameter description see `matplotlib.rcsetup.ValidateInStrings`.

Other Parameters

- **inis** (*tuple*) – Tuple of object types that may pass the check
- **default** (*str*) – The default string to use for an integer (Default: 'rounded')

Methods

instance_check(val)

instance_check(*val*)

class psy_simple.plugin.**DictValValidator**(*key*, *valid*, *validators*, *default*, *ignorecase=False*)

Bases: object

A validation class for formatoptions that expect dictionaries as values

Parameters

- **key** (*str*) – The name of the formatoption (will be used for error handling)
- **valid** (*list of str*) – The valid keys for the dictionary
- **validators** (*func*) – The validation function for the values of the dictionary
- **default** (*object*) – The default value to use if a key from *valid* is given in the provided value
- **ignorecase** (*bool*) – Whether the case of the keys should be ignored

class psy_simple.plugin.**LineWidthValidator**(*key*, *valid*, *ignorecase=False*)
Bases: matplotlib.rcsetup.ValidateInStrings

valid is a list of legal strings

class psy_simple.plugin.**TicksValidator**(*key*, *valid*, *ignorecase=False*)
Bases: matplotlib.rcsetup.ValidateInStrings

valid is a list of legal strings

class psy_simple.plugin.**ValidateList**(*dtype=None*, *length=None*, *listtype=<class 'list'>*)
Bases: object

Validate a list of the specified *dtype*

Parameters

- **dtype** (*object*) – A datatype (e.g. float) that shall be used for the conversion
- **length** (*int*) – The expected length of the list
- **listtype** (*type*) – The type to use for creating the list. Should accept any iterable

Attributes

None	data type (e.g. float) used for the conversion

dtype = None
data type (e.g. float) used for the conversion

psy_simple.plugin.**get_versions**(*requirements=True*)

psy_simple.plugin.**patch_prior_1_0**(*plotter_d*, *versions*)
Patch psy_simple plotters for versions smaller than 1.0

Before psyplot 1.0.0, the plotters in the psy_simple package where part of the psyplot.plotter.simple module. This has to be corrected

psy_simple.plugin.**patches = {('psyplot.plotter.simple', 'BarPlotter'): <function patch_pr**
patches to apply when loading a project

psy_simple.plugin.**rcParams**
the RcParams for the psy-simple plugin

psy_simple.plugin.**try_and_error**(**funcs*)
Apply multiple validation functions

Parameters ***funcs** – Validation functions to test

Returns

Return type function

`psy_simple.plugin.`**`validate_alpha`**(*val*)
Validate an alpha value between 0 and 1

`psy_simple.plugin.`**`validate_axiscolor`**(*value*)
Validate a dictionary containing axiscolor definitions

Parameters **`value`** (*`dict`*) – see `psyplot.plotter.baseplotter.axiscolor`

Returns

Return type dict

Raises `ValueError`

`psy_simple.plugin.`**`validate_cbarpos`**(*value*)
Validate a colorbar position

Parameters **`value`** (*`bool or str`*) – A string can be a combination of 'sh|sv|fl|fr|ft|fb|b|r'

Returns list of strings with possible colorbar positions

Return type list

Raises `ValueError`

`psy_simple.plugin.`**`validate_cmap`**(*val*)
Validate a colormap

Parameters **`val`** (str or `mpl.colors.Colormap`) –

Returns

Return type str or `mpl.colors.Colormap`

Raises `ValueError`

`psy_simple.plugin.`**`validate_cmaps`**(*cmaps*)
Validate a dictionary of color lists

Parameters **`cmaps`** (*`dict`*) – a mapping from a colormap name to a list of colors

Raises `ValueError` – If one of the values in *cmaps* is not a color list

Notes

For all items (listname, list) in *cmaps*, the reversed list is automatically inserted with the `listname + '_r'` key.

`psy_simple.plugin.`**`validate_dataarray`**(*val*)

`psy_simple.plugin.`**`validate_err_calc`**(*val*)
Validation function for the `psy_simple.plotter.FldmeanPlotter.err_calc` formatoption

`psy_simple.plugin.`**`validate_float`**(*s*)
convert *s* to float or raise

Returns **s converted to a float**

Return type float

Raises `ValueError`

`psy_simple.plugin.`**`validate_fontweight`**`(`*value*`)`

`psy_simple.plugin.`**`validate_iter`**`(`*value*`)`
Validate that the given value is an iterable

`psy_simple.plugin.`**`validate_legend`**`(`*value*`)`

`psy_simple.plugin.`**`validate_limits`**`(`*value*`)`

`psy_simple.plugin.`**`validate_lineplot`**`(`*value*`)`
Validate the value for the LinePlotter.plot formatoption

Parameters **`value`** (*`None, str or list with mixture of both`*) – The value to validate

`psy_simple.plugin.`**`validate_marker`**`(`*val*`)`
Does not really make a validation because markers can be quite of different types

`psy_simple.plugin.`**`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`

`psy_simple.plugin.`**`validate_str`**`(`*s*`)`
Validate a string

Parameters **`s`** (*`str`*) –

Returns

Return type str

Raises `ValueError`

`psy_simple.plugin.`**`validate_sym_lims`**`(`*val*`)`

`psy_simple.plugin.`**`validate_text`**`(`*value*`)`
Validate a text formatoption

Parameters **`value`** (see `psyplot.plotter.labelplotter.text`) –

Raises `ValueError`

`psy_simple.plugin.`**`validate_ticklabels`**`(`*value*`)`

psy_simple.version module

1.5 Changelog

1.5.1 v1.1.0

Added

- Changelog
- `interp_bounds` formatoption for the `plot2d` plot method (see the docs)

- Added the `fldmean` plot method that can be used to directly calculate and plot the mean over the x- and y-dimensions

Changed

- The xlim and ylim formatoptions now consider inverted x- and y-axes

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Bibliography

[statsmodels] http://statsmodels.sourceforge.net/

[statsmodels] http://statsmodels.sourceforge.net/

Python Module Index

p

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Y