
psy-simple Documentation

Release 1.1.0

Philipp Sommer

Apr 09, 2018

1	Documentation	3
1.1	Installation	3
1.1.1	How to install	3
	Installation using conda	3
	Installation using pip	3
1.1.2	Running the tests	3
1.2	psypilot plot methods	4
1.2.1	psypilot.project.plot.lineplot	4
1.2.2	psypilot.project.plot.vector	5
1.2.3	psypilot.project.plot.violinplot	6
1.2.4	psypilot.project.plot.plot2d	7
1.2.5	psypilot.project.plot.combined	8
1.2.6	psypilot.project.plot.density	9
1.2.7	psypilot.project.plot.barplot	10
1.2.8	psypilot.project.plot.fldmean	11
1.3	Example Gallery	12
1.3.1	Line plot demo	12
1.3.2	Bar plot demo	15
1.3.3	2D plots	20
1.3.4	Violin plot demo	23
1.3.5	Vector plot	26
1.4	API Reference	30
1.4.1	Subpackages	30
	psypilot.widgets package	30
	Submodules	30
	psypilot.widgets.colors module	30
	psypilot.widgets.texts module	30
1.4.2	Submodules	30
	psypilot.base module	30
	Possible types	32
	Possible types	32
	Possible types	32
	Possible types	33
	Possible types	33
	Possible types	33
	Possible types	34

Notes	34
Possible types	34
Possible types	34
Possible types	35
Possible types	35
Possible types	36
Notes	36
Possible types	36
Possible types	36
Possible types	37
Possible types	37
Possible types	38
Possible types	39
Notes	39
Possible types	40
Possible types	41
Possible types	42
Possible types	43
Possible types	43
Possible types	45
Notes	46
Possible types	48
Possible types	49
Notes	49
Possible types	51
Notes	51
Possible types	52
Possible types	52
Possible types	52
Possible types	52
Possible types	53
Possible types	53
Notes	54
Possible types	54
Possible types	54
Possible types	54
Possible types	55
Possible types	55
psy_simple.colors module	58
Notes	58
Notes	59
Notes	59
Notes	61
References	62
psy_simple.plotters module	62
Possible types	65
Possible types	67
Possible types	68
Possible types	70
Notes	70
Possible types	71
Notes	71
Possible types	72
Possible types	73
Possible types	76

Possible types	76
Possible types	77
Possible types	78
Possible types	78
Possible types	78
Possible types	79
Possible types	80
Possible types	80
Possible types	80
Possible types	81
Possible types	81
Possible types	81
Possible types	81
Possible types	82
Possible types	82
Possible types	82
Notes	82
Possible types	83
Possible types	83
Possible types	83
Possible types	83
Possible types	84
Possible types	84
Possible types	85
Possible types	85
Notes	86
Possible types	86
Possible types	86
Possible types	86
Possible types	87
Possible types	87
Possible types	88
Notes	88
Possible types	88
Notes	88
Possible types	89
Possible types	89
Possible types	89
Possible types	90
Possible types	90
Possible types	91
Possible types	91
Possible types	92
Possible types	93
Possible types	93
Possible types	93
Possible types	94
Possible types	94
Possible types	94
Possible types	95
Possible types	97
Possible types	97
Possible types	99
Possible types	100

Possible types	101
Possible types	103
Possible types	104
Possible types	104
Possible types	105
Possible types	105
Possible types	105
Possible types	105
Possible types	106
Possible types	106
Possible types	106
Possible types	107
Possible types	107
Possible types	107
Possible types	108
Possible types	108
Possible types	108
Possible types	109
Possible types	111
Possible types	111
Notes	111
Possible types	111
Possible types	112
Possible types	113
Possible types	113
Possible types	114
Notes	114
Possible types	114
Possible types	115
Possible types	115
Possible types	115
Possible types	115
Possible types	115
Possible types	116
Possible types	116
Possible types	116
Possible types	117
Possible types	117
Possible types	117
Possible types	117
Possible types	117
Possible types	118
Possible types	118
Possible types	119
Possible types	119
Possible types	121
Possible types	122
Possible types	124
Possible types	125
Possible types	126
Possible types	127
Possible types	128
Possible types	129
Possible types	133
Possible types	136
Possible types	136

Notes	136
Possible types	137
Possible types	137
Possible types	137
Possible types	138
Possible types	139
Possible types	139
Possible types	139
Possible types	140
Possible types	140
Possible types	140
Possible types	140
Possible types	141
Possible types	141
Possible types	141
Possible types	142
Possible types	142
Possible types	143
Possible types	143
Possible types	144
Possible types	144
Possible types	144
Possible types	144
Possible types	144
Possible types	144
Possible types	145
Possible types	145
Possible types	149
Notes	149
Possible types	149
Possible types	150
Possible types	150
Possible types	151
Possible types	151
Possible types	151
Possible types	152
Possible types	152
Possible types	152
Possible types	153
Possible types	153
Possible types	153
Possible types	154
Possible types	154
Possible types	155
Possible types	155
Possible types	155
Possible types	155
Possible types	156
Possible types	156
Possible types	156
Possible types	156
Possible types	157
Possible types	157
Possible types	157
Possible types	158

Possible types	158
Possible types	158
Possible types	158
Possible types	159
Notes	159
Possible types	159
Possible types	159
Possible types	160
Possible types	160
Possible types	160
Possible types	160
Possible types	160
Possible types	161
Possible types	162
Notes	162
Possible types	162
Possible types	162
Possible types	163
Possible types	163
Possible types	163
Possible types	164
Possible types	164
Possible types	165
Possible types	165
Possible types	165
Possible types	166
Possible types	166
Notes	167
Possible types	167
Notes	167
Possible types	168
Possible types	168
Possible types	168
Possible types	169
Possible types	169
Possible types	169
Possible types	170
Possible types	170
Possible types	171
Possible types	171
Possible types	171
Possible types	171
Possible types	172
Possible types	173
Possible types	173
Possible types	173
Possible types	174
Possible types	175
Possible types	175
Notes	175
Possible types	175
Possible types	176
Possible types	176
Possible types	176
Possible types	176
Possible types	177

Possible types	177
Possible types	178
Possible types	179
Possible types	181
Possible types	182
Possible types	183
Notes	183
Possible types	187
Possible types	187
Possible types	188
References	188
Possible types	188
Possible types	188
Possible types	189
Notes	189
Possible types	189
Notes	190
Possible types	190
Possible types	191
Possible types	191
Possible types	191
Possible types	191
Possible types	192
Possible types	192
Possible types	192
Possible types	192
Possible types	192
Possible types	193
Possible types	193
Possible types	193
Possible types	193
Possible types	193
Possible types	194
Possible types	194
Possible types	194
Possible types	194
Possible types	195
Possible types	195
Possible types	195
Possible types	195
Possible types	195
Possible types	196
Notes	196
Possible types	196
Possible types	197
Possible types	197
Possible types	197
Possible types	197
Possible types	197
Possible types	198
Possible types	198
Possible types	199
Notes	199
Possible types	199
Possible types	200
Possible types	200
Possible types	200
Possible types	200
Possible types	201
Possible types	201
Possible types	202

Possible types	202
Notes	202
Possible types	203
Notes	203
Possible types	204
Possible types	204
Possible types	204
Possible types	204
Possible types	205
Possible types	205
Possible types	206
Possible types	206
Possible types	206
Possible types	207
Possible types	208
Possible types	208
Possible types	208
Possible types	209
Possible types	210
Possible types	210
Possible types	210
Possible types	210
Possible types	211
Possible types	211
Possible types	211
Possible types	213
Possible types	214
Possible types	216
Possible types	217
Possible types	220
Possible types	221
Possible types	221
Possible types	221
Possible types	222
Possible types	222
Possible types	222
Possible types	222
Possible types	222
Possible types	222
Possible types	223
Possible types	223
Notes	223
Possible types	224
Possible types	224
Possible types	224
Possible types	224
Possible types	225
Possible types	225
Possible types	226
Possible types	226
Notes	227
Possible types	227
Possible types	227
Possible types	227
Possible types	228

Possible types	228
Possible types	229
Possible types	229
Possible types	230
Notes	230
Possible types	230
Notes	230
Possible types	231
Possible types	231
Possible types	231
Possible types	232
Possible types	232
Possible types	233
Possible types	233
Possible types	233
Possible types	234
Possible types	235
Possible types	235
Possible types	235
Possible types	236
Possible types	237
Possible types	237
Possible types	237
Possible types	237
Possible types	238
Possible types	238
Possible types	238
Possible types	239
Possible types	239
Possible types	239
Notes	240
Possible types	241
Notes	241
Possible types	242
Notes	243
Possible types	243
Possible types	244
Possible types	245
Possible types	246
Possible types	247
Possible types	249
Possible types	250
Possible types	252
Possible types	253
Possible types	254
Possible types	255
Possible types	258
Possible types	259
Possible types	260
Possible types	260
Possible types	260
Possible types	260
Possible types	261
Possible types	261

Possible types	261
Possible types	261
Possible types	262
Possible types	262
Possible types	262
Possible types	263
Notes	263
Possible types	263
Possible types	263
Possible types	263
Possible types	264
Possible types	264
Possible types	264
Possible types	265
Possible types	266
Notes	266
Possible types	266
Possible types	266
Possible types	266
Possible types	267
Possible types	268
Possible types	268
Possible types	269
Possible types	269
Notes	269
Possible types	269
Notes	270
Possible types	270
Possible types	270
Possible types	270
Possible types	271
Possible types	271
Possible types	272
Possible types	272
Possible types	272
Possible types	273
Possible types	274
Possible types	274
Possible types	274
Possible types	275
Possible types	276
Possible types	276
Possible types	276
Possible types	277
Possible types	277
Possible types	278
Possible types	279
Possible types	280
Possible types	281
Possible types	282
Possible types	284
Possible types	286
References	286
Possible types	288

Possible types	289
Possible types	290
Possible types	290
Possible types	291
Possible types	291
Possible types	291
Possible types	291
Possible types	291
Possible types	291
Possible types	292
Possible types	294
Possible types	295
Possible types	295
Possible types	296
Possible types	296
Possible types	296
Possible types	297
Possible types	297
Possible types	297
Possible types	298
Possible types	298
Possible types	298
Possible types	298
Possible types	299
Possible types	299
Possible types	300
Possible types	300
Possible types	301
Possible types	303
Possible types	304
Possible types	304
Possible types	304
Possible types	304
Possible types	304
Possible types	305
Possible types	306
Possible types	307
Possible types	308
Possible types	308
Possible types	309
Possible types	309
Possible types	309
Possible types	310
Possible types	310
Possible types	310
Possible types	310
Possible types	310
Possible types	311
Possible types	311
Possible types	311
Possible types	311
Possible types	312
Possible types	312
Possible types	312
Possible types	312
Possible types	313

Notes	313
Possible types	314
Possible types	314
Possible types	314
Possible types	314
Possible types	315
Possible types	315
Possible types	316
Possible types	316
Notes	316
Possible types	317
Possible types	317
Possible types	317
Possible types	318
Possible types	318
Possible types	319
Possible types	319
Possible types	320
Notes	320
Possible types	320
Notes	320
Possible types	321
Possible types	321
Possible types	321
Possible types	322
Possible types	322
Possible types	323
Possible types	323
Possible types	323
Possible types	324
Possible types	324
Possible types	324
Possible types	324
Possible types	325
Possible types	325
Possible types	328
Notes	328
Possible types	329
Possible types	330
Notes	330
Possible types	330
Possible types	331
Possible types	331
Possible types	331
Possible types	332
Possible types	332
Possible types	333
Possible types	333
Possible types	333
Possible types	333
Possible types	333
Possible types	334
Notes	334
Possible types	334

Possible types	335
Possible types	335
Possible types	335
Possible types	335
Possible types	336
Possible types	336
Possible types	337
Notes	337
Possible types	337
Possible types	338
Possible types	338
Possible types	338
Possible types	339
Possible types	339
Possible types	340
Possible types	340
Possible types	341
Possible types	341
Possible types	341
Possible types	341
Possible types	341
Possible types	343
Possible types	343
Possible types	343
Possible types	343
Possible types	344
Possible types	345
Possible types	345
Possible types	349
Possible types	349
Possible types	349
Possible types	351
Possible types	351
Possible types	352
Possible types	353
Possible types	353
Possible types	353
Possible types	354
Possible types	354
Possible types	354
Possible types	355
Possible types	355
Possible types	355
Possible types	355
Possible types	355
Possible types	356
Possible types	356
Possible types	356
Possible types	357
Possible types	357
Possible types	357
Possible types	358
Notes	358
Possible types	358
Possible types	358

Possible types	358
Possible types	359
Possible types	359
Possible types	359
Possible types	360
Possible types	361
Notes	361
Possible types	361
Possible types	361
Possible types	362
Possible types	362
Possible types	363
Possible types	363
Possible types	364
Possible types	364
Notes	364
Possible types	365
Notes	365
Possible types	365
Possible types	366
Possible types	366
Possible types	366
Possible types	366
Possible types	367
Possible types	367
Possible types	367
Possible types	367
Possible types	368
Possible types	368
Possible types	368
Possible types	368
Possible types	368
Notes	369
Possible types	369
Notes	369
Possible types	369
Possible types	369
Possible types	370
Possible types	370
Possible types	370
Possible types	370
Possible types	371
Possible types	372
Possible types	373
Possible types	374
Possible types	376
Possible types	377
Possible types	378
Possible types	380
Possible types	382
Possible types	382
Possible types	384
Possible types	385
Possible types	386
Possible types	387
Possible types	388
Possible types	389

Possible types	391
Possible types	394
Possible types	394
Possible types	395
Possible types	395
Possible types	396
Possible types	397
Possible types	397
Possible types	398
Possible types	398
Possible types	399
Possible types	399
Possible types	399
Possible types	399
Possible types	400
Notes	400
Possible types	400
Possible types	400
Possible types	401
Possible types	401
Possible types	401
Possible types	401
Possible types	401
Possible types	402
Possible types	403
Notes	403
Possible types	403
Possible types	403
Possible types	404
Possible types	404
Possible types	405
Possible types	405
Possible types	406
Possible types	406
Notes	406
Possible types	407
Notes	407
Possible types	407
Possible types	408
Possible types	408
Possible types	408
Possible types	408
Possible types	409
Possible types	409
Possible types	410
Possible types	410
Possible types	410
Possible types	410
Possible types	411
Possible types	411
Possible types	413
Possible types	414
Possible types	415
Possible types	416
Possible types	417

	Possible types	418
	Possible types	419
	Possible types	420
	Possible types	422
	Possible types	424
	Possible types	425
	Possible types	425
	Possible types	425
	Possible types	426
	Possible types	426
	Possible types	426
	Possible types	427
	Possible types	427
	Possible types	427
	Possible types	428
	Possible types	429
	Possible types	429
	Possible types	430
	Possible types	430
	Possible types	430
	Possible types	431
	Possible types	432
	Possible types	433
	Possible types	434
	Possible types	435
	Possible types	436
	Possible types	438
	Possible types	438
	Possible types	439
	Possible types	440
	Possible types	442
	Possible types	443
	Possible types	445
	psy_simple.plugin module	446
	Notes	449
	psy_simple.version module	450
1.5	Changelog	450
1.5.1	v1.1.0	450
	Added	450
	Changed	451
2	Indices and tables	453
	Bibliography	455
	Python Module Index	457

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.

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

<i>axiscolor</i>	<i>color</i>	<i>coord</i>	<i>error</i>
<i>erroralpha</i>	<i>figtitle</i>	<i>figtitleprops</i>	<i>figtitlesize</i>
<i>figtitleweight</i>	<i>grid</i>	<i>labelprops</i>	<i>labelsizes</i>
<i>labelweight</i>	<i>legend</i>	<i>legendlabels</i>	<i>linewidth</i>
<i>marker</i>	<i>markersize</i>	<i>maskbetween</i>	<i>maskgeq</i>
<i>maskgreater</i>	<i>maskleq</i>	<i>maskless</i>	<i>plot</i>
<i>post</i>	<i>post_timing</i>	<i>sym_lims</i>	<i>text</i>
<i>ticksize</i>	<i>tickweight</i>	<i>tight</i>	<i>title</i>
<i>titleprops</i>	<i>titlesize</i>	<i>titleweight</i>	<i>transpose</i>
<i>xlabel</i>	<i>xlim</i>	<i>xrotation</i>	<i>xticklabels</i>
<i>xtickprops</i>	<i>xticks</i>	<i>ylabel</i>	<i>ylim</i>
<i>yrotation</i>	<i>yticklabels</i>	<i>ytickprops</i>	<i>yticks</i>

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
# formatoptions
>>> 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

<i>arrowsize</i>	<i>arrowstyle</i>	<i>axiscolor</i>	<i>bounds</i>
<i>cbar</i>	<i>cbarspacing</i>	<i>clabel</i>	<i>clabelprops</i>
<i>clabelsize</i>	<i>clabelweight</i>	<i>cmap</i>	<i>color</i>
<i>cticklabels</i>	<i>ctickprops</i>	<i>cticks</i>	<i>cticksize</i>
<i>ctickweight</i>	<i>datagrid</i>	<i>density</i>	<i>extend</i>
<i>figtitle</i>	<i>figtitleprops</i>	<i>figtitlesize</i>	<i>figtitleweight</i>
<i>grid</i>	<i>labelprops</i>	<i>labelsize</i>	<i>labelweight</i>
<i>linewidth</i>	<i>maskbetween</i>	<i>maskgeq</i>	<i>maskgreater</i>
<i>maskleq</i>	<i>maskless</i>	<i>plot</i>	<i>post</i>
<i>post_timing</i>	<i>sym_lims</i>	<i>text</i>	<i>ticksize</i>
<i>tickweight</i>	<i>tight</i>	<i>title</i>	<i>titleprops</i>
<i>titlesize</i>	<i>titleweight</i>	<i>transpose</i>	<i>xlabel</i>
<i>xlim</i>	<i>xrotation</i>	<i>xticklabels</i>	<i>xtickprops</i>
<i>xticks</i>	<i>ylabel</i>	<i>ylim</i>	<i>yrotation</i>
<i>yticklabels</i>	<i>ytickprops</i>	<i>yticks</i>	

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
# formatoptions
>>> 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

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

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
# formatoptions
>>> 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

<i>axiscolor</i>	<i>bounds</i>	<i>cbar</i>	<i>cbarspacing</i>
<i>clabel</i>	<i>clabelprops</i>	<i>clabelsize</i>	<i>clabelweight</i>
<i>cmap</i>	<i>cticklabels</i>	<i>ctickprops</i>	<i>cticks</i>
<i>cticksiz</i>	<i>ctickweight</i>	<i>datagrid</i>	<i>extend</i>
<i>figtitle</i>	<i>figtitleprops</i>	<i>figtitlesize</i>	<i>figtitleweight</i>
<i>grid</i>	<i>interp_bounds</i>	<i>labelprops</i>	<i>labelsize</i>
<i>labelweight</i>	<i>levels</i>	<i>maskbetween</i>	<i>maskgeq</i>
<i>maskgreater</i>	<i>maskleq</i>	<i>maskless</i>	<i>miss_color</i>
<i>plot</i>	<i>post</i>	<i>post_timing</i>	<i>sym_lims</i>
<i>text</i>	<i>ticksiz</i>	<i>tickweight</i>	<i>tight</i>
<i>title</i>	<i>titleprops</i>	<i>titlesize</i>	<i>titleweight</i>
<i>transpose</i>	<i>xlabel</i>	<i>xlim</i>	<i>xrotation</i>
<i>xticklabels</i>	<i>xtickprops</i>	<i>xticks</i>	<i>ylabel</i>
<i>ylim</i>	<i>yrotation</i>	<i>yticklabels</i>	<i>ytickprops</i>
<i>yticks</i>			

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
# formatoptions
>>> 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

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

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
# formatoptions
>>> 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

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

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
# formatoptions
>>> 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

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

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
# formatoptions
>>> 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

<i>axiscolor</i>	<i>color</i>	<i>coord</i>	<i>err_calc</i>
<i>error</i>	<i>erroralpha</i>	<i>figtitle</i>	<i>figtitleprops</i>
<i>figtitlesize</i>	<i>figtitleweight</i>	<i>grid</i>	<i>labelprops</i>
<i>labelsiz</i>	<i>labelweight</i>	<i>legend</i>	<i>legendlabels</i>
<i>linewidth</i>	<i>marker</i>	<i>markersize</i>	<i>maskbetween</i>
<i>maskgeq</i>	<i>maskgreater</i>	<i>maskleq</i>	<i>maskless</i>
<i>mean</i>	<i>plot</i>	<i>post</i>	<i>post_timing</i>
<i>sym_lim</i>	<i>text</i>	<i>ticksize</i>	<i>tickweight</i>
<i>tight</i>	<i>title</i>	<i>titleprops</i>	<i>titlesize</i>
<i>titleweight</i>	<i>transpose</i>	<i>xlabel</i>	<i>xlim</i>
<i>xrotation</i>	<i>xticklabels</i>	<i>xtickprops</i>	<i>xticks</i>
<i>ylabel</i>	<i>ylim</i>	<i>yrotation</i>	<i>yticklabels</i>
<i>ytickprops</i>	<i>yticks</i>		

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
# formatoptions
>>> 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)

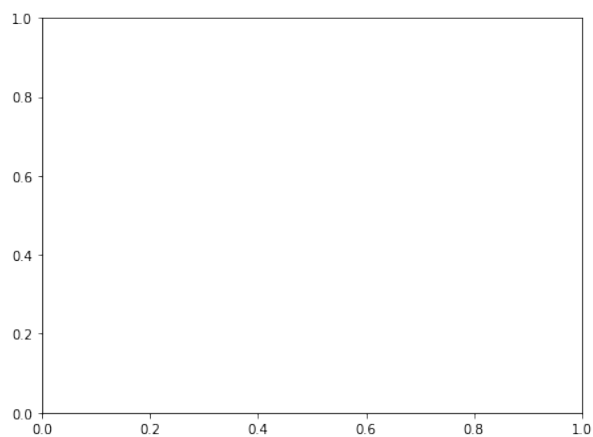
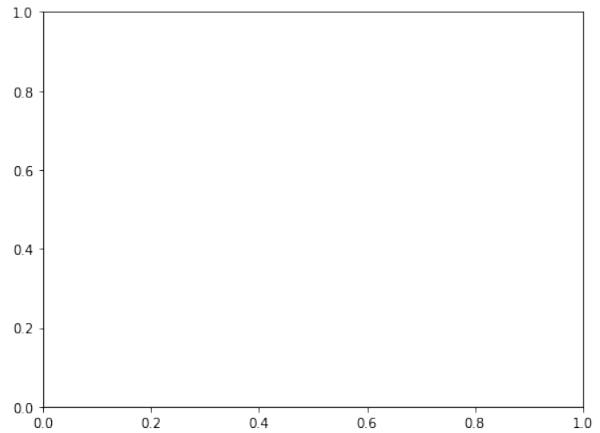
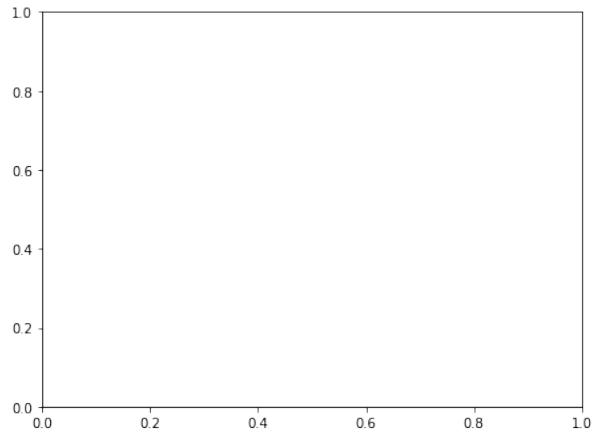
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↳ packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in <module>()
5512
5513
-> 5514 class FldmeanPlotter(LinePlotter):
5515
5516     _rcparams_string = ["plotter.fldmean."]

~/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)

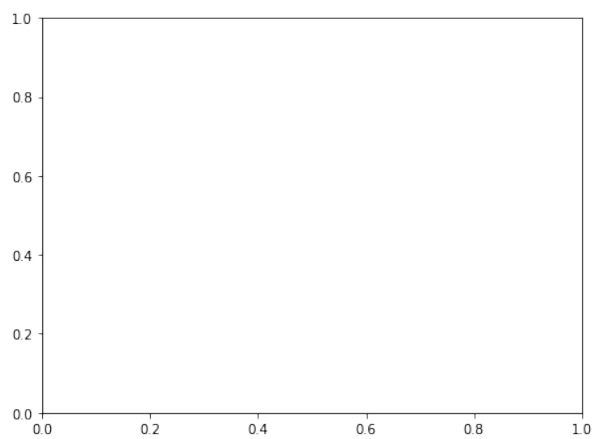
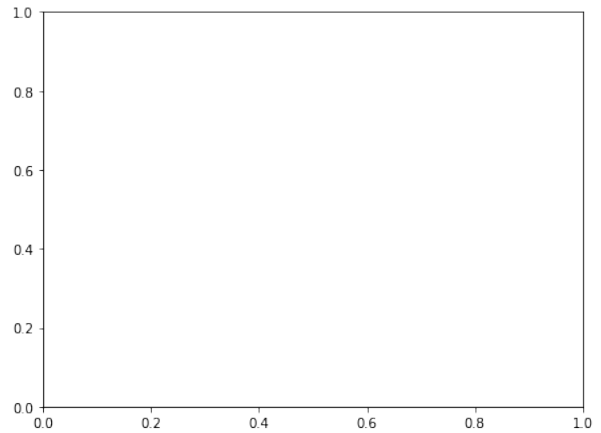
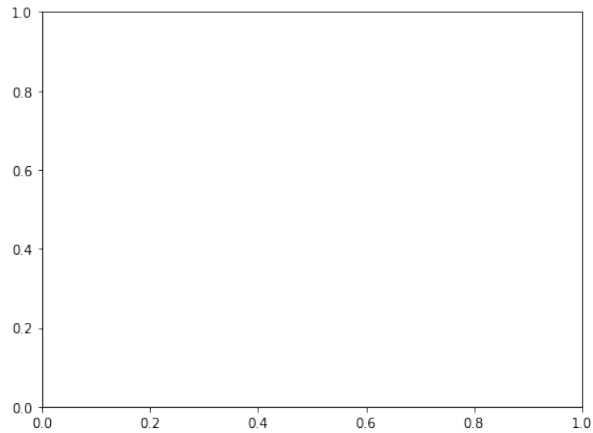
```
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/
↳ importlib/_bootstrap.py in _call_with_frames_removed(f, *args, **kwds)

~/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'])

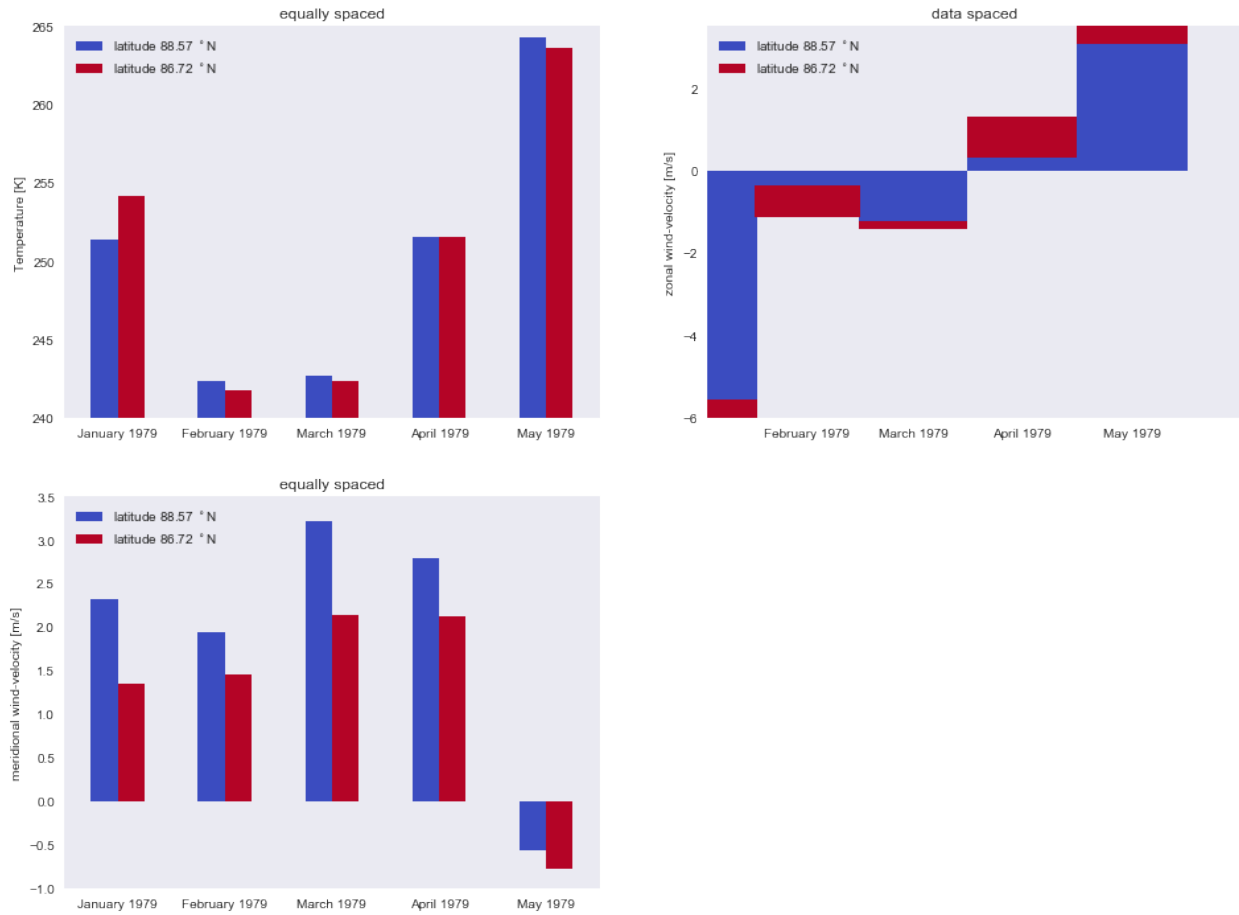
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↳ packages/psyplot/plotter.py in __init__(self, key, plotter, index_in_list, **kwargs)
429         raise TypeError(
430             '%s.__init__() got an unexpected keyword argument %r' % (
--> 431                 self.__class__.__name__,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

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



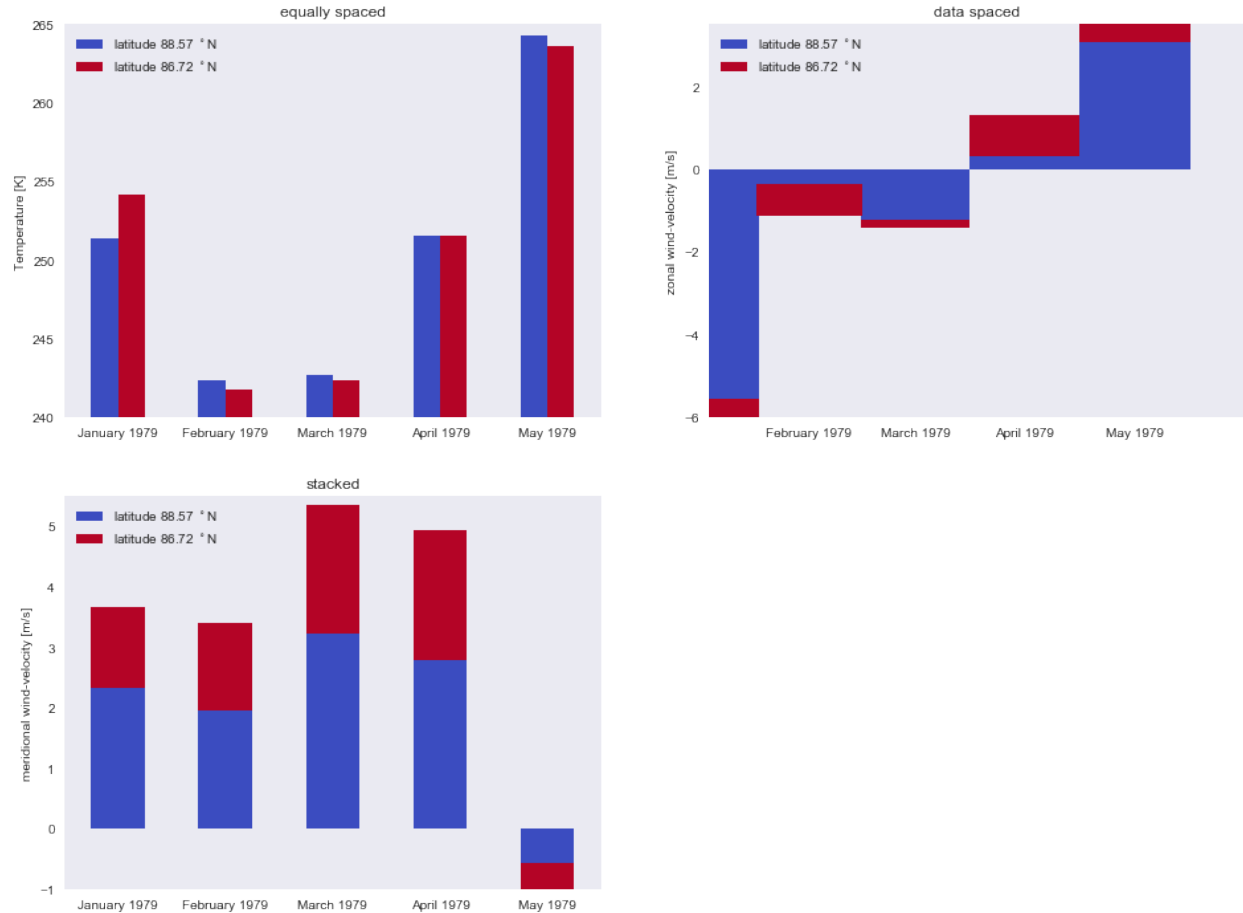
The default is that all bars have the same width. You can however change that by setting the `widths` keyword to `data`

```
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 matplotlib's `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/psypilot/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/psypilot/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, **kwargs)

~/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."]

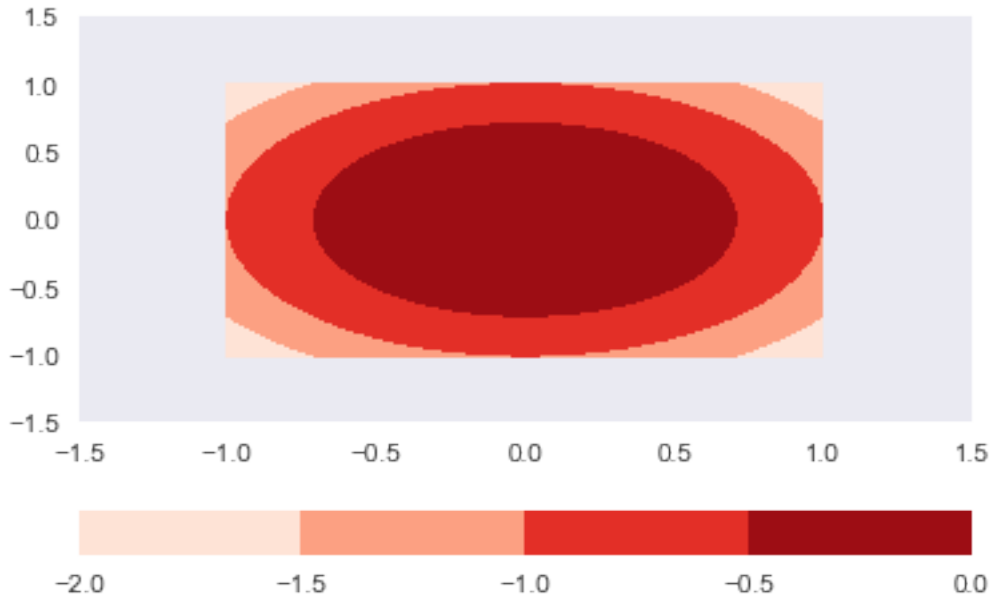
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↳packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in FldmeanPlotter()
    5523     # We reimplement the masking formatoption to make sure, that they are
    5524     # called after the mean calculation
-> 5525     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	cticksiz
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)

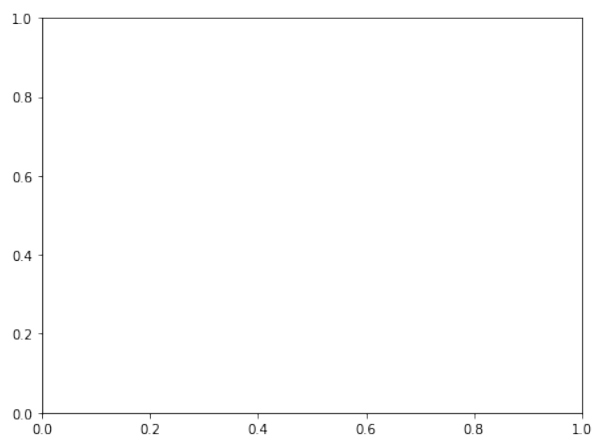
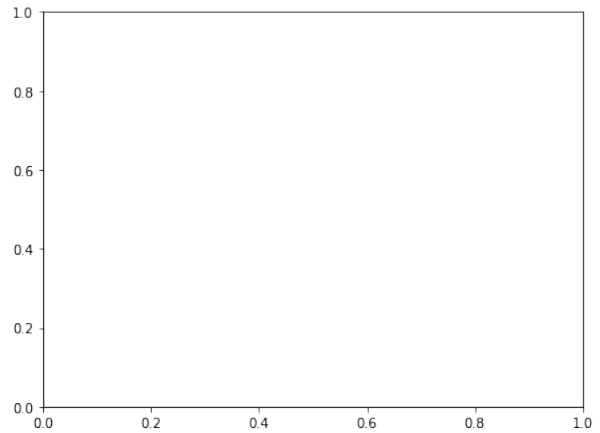
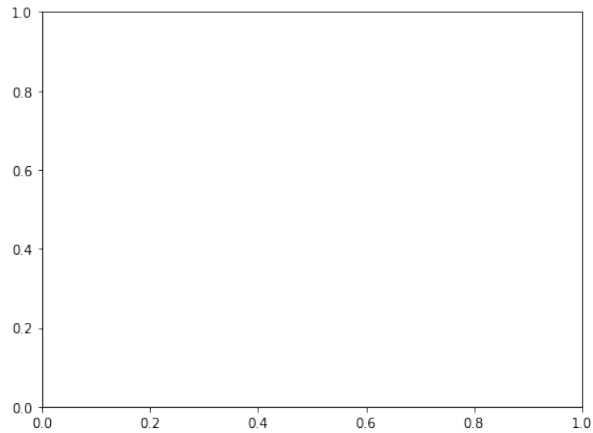
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↳packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in <module>()
5512
5513
-> 5514 class FldmeanPlotter(LinePlotter):
5515
5516     _rcparams_string = ["plotter.fldmean."]

~/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 matplotlib's `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/psypoint/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/psypoint/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."]

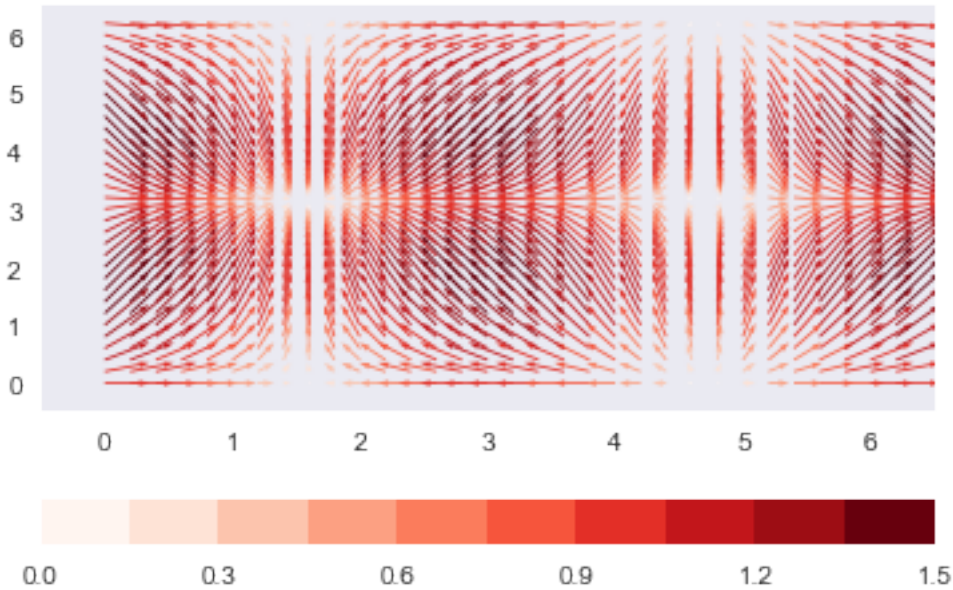
~/checkouts/readthedocs.org/user_builds/psy-simple/conda/latest/lib/python3.5/site-
↳packages/psy_simple-1.1.0-py3.5.egg/psy_simple/plotters.py in FldmeanPlotter()
5523     # We reimplement the masking formatoption to make sure, that they are
5524     # called after the mean calculation
-> 5525     maskgeq = 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'
```

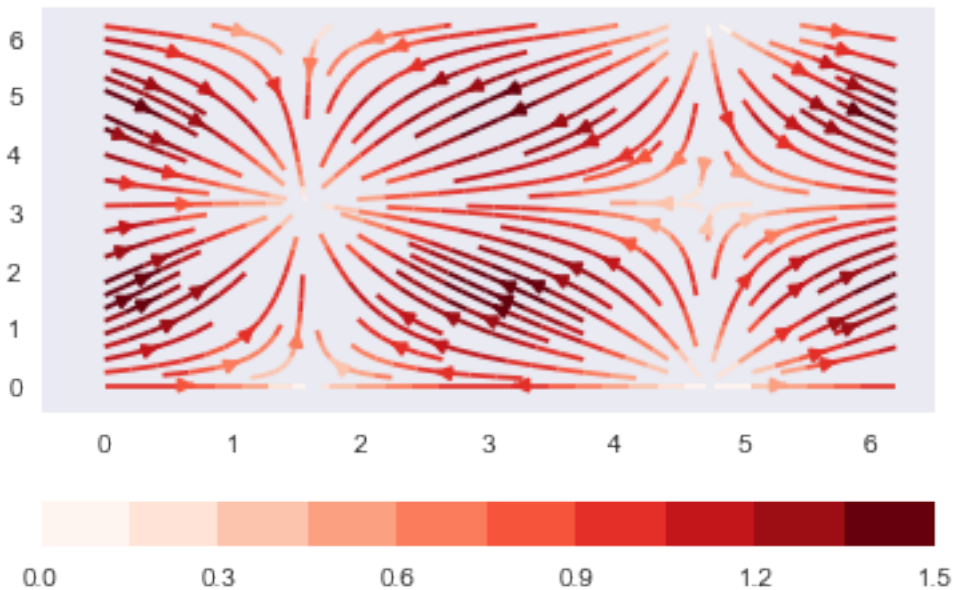
You can also apply a colormap to the vectors via the `colors` formatoption. This can be one on `'u'` (to use the x-direction), `'v'` (to use the `'y'`-direction) or `'absolute'` to use the absolute length of the arrows.

```
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 pyplot plugin for simple visualizations

This package contains the basic plotters for simple interactive visualization tasks with the pyplot 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

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

Formatoption classes

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

Classes

<code>TextBase</code>	Abstract base class for formatoptions that provides a replace method
-----------------------	--

Functions

<code>label_props</code> (base[, label_name, children, ...])	Function that returns a Formatoption class for modifying the fontsize
<code>label_size</code> (base[, label_name, children, ...])	Function that returns a Formatoption class for modifying the fontsize
<code>label_weight</code> (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

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

Axes formatoptions

<i>tight</i>	Automatically adjust the plots.
Label formatoptions	
<i>figtitle</i>	Plot a figure title
<i>figtitleprops</i>	Properties of the figure title
<i>figtitlesize</i>	Set the size of the figure title
<i>figtitleweight</i>	Set the fontweight of the figure title
<i>text</i>	Add text anywhere on the plot
<i>title</i>	Show the title
<i>titleprops</i>	Properties of the title
<i>titlesize</i>	Set the size of the title
<i>titleweight</i>	Set the fontweight of the title
Post processing formatoptions	
<i>post</i>	Apply your own postprocessing script
<i>post_timing</i>	Determine when to run the <code>post</code> 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 `psypplot.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 appearance (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 `psypplot.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

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

Attributes

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

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 at 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, addi-  
                                tional_children=[], additional_dependencies=[],  
                                **kwargs)
```

Bases: `psy_simple.base.ValueMaskBase`

Mask data points between two numbers

Possible types

Methods

<code>mask_func</code> (data, value)	The masking function that is called
--------------------------------------	-------------------------------------

Attributes

<code>name</code>	<code>str(object='') -> str</code>
-------------------	---------------------------------------

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

<code>mask_func</code> (<i>data</i> , <i>value</i>)	The masking function that is called
---	-------------------------------------

Attributes

<i>name</i>	<code>str(object='') -> str</code>
-------------	---------------------------------------

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

<code>mask_func(data, value)</code>	The masking function that is called
-------------------------------------	-------------------------------------

Attributes

<code>name</code>	<code>str(object='') -> str</code>
-------------------	---------------------------------------

`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

<code>mask_func(data, value)</code>	The masking function that is called
-------------------------------------	-------------------------------------

Attributes

<code>name</code>	<code>str(object='') -> str</code>
-------------------	---------------------------------------

`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

<code>mask_func(data, value)</code>	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

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

Attributes

<code>name</code>	<code>str(object='') -> str</code>
<code>transform</code>	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. $0 \leq x, y \leq 1$ 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 appearance (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 (*fmo*, ***kwargs*)

Share the settings of this formatoption with other data objects

Parameters

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

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**

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

Methods

<code>get_enhanced_attrs(*args, **kwargs)</code>	
<code>get_fig_data_attrs([delimiter])</code>	Join the data attributes with other plotters in the project
<code>get_fmt_widget(parent, project)</code>	Create a combobox with the attributes
<code>replace(s, data[, attrs])</code>	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

<code>group</code>	<code>str(object='') -> str</code>
<code>name</code>	<code>str(object='') -> str</code>

Methods

<code>update(value)</code>	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

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

Attributes

<code>name</code>	<code>str(object='') -> str</code>
-------------------	---------------------------------------

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

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

Post processing formatoptions

<code>post</code>	Apply your own postprocessing script
<code>post_timing</code>	Determine when to run the <code>post</code> 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 appearance (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

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

Methods

<code>mask_func()</code>	The masking function that is called
<code>update(value)</code>	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 fontsize

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 fontsize

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

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

Functions

<code>get_cmap</code> (name[, lut])	Returns the specified colormap.
<code>show_colormaps</code> ([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.streampplot()` 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 matplotlib's 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::matplotlib.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 on¹. If None, use the Qt implementation if it is running in the pyplot 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 `matplotlib.figure.Figure`

References

psy_simple.plotters module

Formatoption classes

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

Continued on next page

Table 37 – continued from previous page

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

Continued on next page

Table 37 – continued from previous page

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

Plotter classes

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

Functions

<i>format_coord_func</i> (ax, ref)	Create a function that can replace the
<i>round_to_05</i> (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

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

Methods

<code>diff(value)</code>	Checks whether the given value differs from what is currently set
<code>get_alternative_coord(da, i)</code>	
<code>replace_coord(i)</code>	Replace the coordinate for the data array at the given position
<code>update(value)</code>	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,          in-
                                                  dex_in_list=None,          addi-
                                                  tional_children=[],          addi-
                                                  tional_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

<code>use_raw_data</code>	<code>bool(x) -> bool</code>
---------------------------	---------------------------------

- *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

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

Methods

<code>update(value)</code>	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** (*psysimple.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 *psysimple.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: psy_simple.plotter.Formatoption
```

Change the style of the arrows

Possible types

Attributes

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

Methods

<i>update</i> (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** (*psysplot.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 `psysplot.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

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

Methods

<i>initialize_plot</i> (<i>value</i>)	Method that is called when the plot is made the first time
<i>update</i> (<i>value</i>)	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

<i>priority</i>	<i>int(x=0) -> integer</i>
-----------------	-------------------------------

Methods

<i>update</i> (<i>value</i>)	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

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

Methods

<i>get_xys</i> (arr)	
<i>make_plot</i> ()	
<i>remove</i> ()	Method to remove the effects of this formatoption
<i>update</i> (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

Miscellaneous formatoptions

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

Data manipulation formatoptions

<i>coord</i>	Use an alternative variable as x-coordinate
--------------	---

Plot formatoptions

<i>plot</i>	Choose how to make the bar plot
-------------	---------------------------------

Label formatoptions

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

Axes formatoptions

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

Continued on next page

Table 57 – continued from previous page

<i>transpose</i>	Switch x- and y-axes
Axis tick formatoptions	
<i>xticklabels</i>	Modify the x-axis ticklabels
<i>yticklabels</i>	Modify the y-axis ticklabels
<i>xrotation</i>	Rotate the x-axis ticks
<i>xtickprops</i>	Specify the x-axis tick parameters
<i>xticks</i>	Modify the x-axis ticks
<i>yrotation</i>	Rotate the y-axis ticks
<i>ytickprops</i>	Specify the y-axis tick parameters
<i>yticks</i>	Modify the y-axis ticks
Color coding formatoptions	
<i>color</i>	Set the color coding
Masking formatoptions	
<i>maskbetween</i>	Mask data points between two numbers
<i>maskgeq</i>	Mask data points greater than or equal to a number
<i>maskgreater</i>	Mask data points greater than a number
<i>maskleq</i>	Mask data points smaller than or equal to a number
<i>maskless</i>	Mask data points smaller than a number
Post processing formatoptions	
<i>post</i>	Apply your own postprocessing script
<i>post_timing</i>	Determine when to run the <code>post</code> 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.

roundsym 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 `psypplot.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. $0 \leq x, y \leq 1$ 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 appearance (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, addi-  
                                     tional_children=[], additional_dependencies=[],  
                                     **kwargs)
```

Bases: `psyplot.plotter.Formatoption`

Specify the widths of the bars

Possible types**Attributes**

<i>priority</i>	<code>int(x=0) -> integer</code>
-----------------	-------------------------------------

Methods

<code>update(value)</code>	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

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

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

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

Continued on next page

Table 67 – continued from previous page

<code>plot</code>	plot Formatoption instance in the plotter
<code>sym_lims</code>	sym_lims Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>widths</code>	widths Formatoption instance in the plotter
<code>xticks</code>	xticks Formatoption instance in the plotter
<code>ylim</code>	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

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

Methods

<i>set_stringformatter(s)</i>

- *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

<code>transpose</code>	transpose Formatoption instance in the plotter
<code>update_after_plot</code>	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

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

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

Label formatoptions

<code>clabel</code>	Show the colorbar label
<code>clabelprops</code>	Properties of the Colorbar label
<code>clabelsize</code>	Set the size of the Colorbar label
<code>clabelweight</code>	Set the fontweight of the Colorbar label

Attributes

<code>convert_radian</code>	Boolean that is <i>True</i> if triangles with units in radian should be
-----------------------------	---

Axis tick formatoptions

<code>cticklabels</code>	Specify the colorbar ticklabels
<code>cticks</code>	Specify the tick locations of the colorbar

Miscellaneous formatoptions

<code>datagrid</code>	Show the grid of the data
-----------------------	---------------------------

Post processing formatoptions

<code>post</code>	Apply your own postprocessing script
<code>post_timing</code>	Determine when to run the <code>post</code> 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 `psypplot.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:

label, 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:

label, 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:

`cticksize`, `ctickprops`, `cticklabels`, `cticks`, `vcticksize`, `vtickprops`, `vticklabels`, `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

<i>arrowsize</i>	Change the size of the arrows
<i>arrowstyle</i>	Change the style of the arrows
<i>density</i>	Change the density of the arrows

Color coding formatoptions

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

Methods

<i>check_data</i> (name, dims, is_unstructured)	A validation method for the data shape
---	--

Axis tick formatoptions

<i>cticks</i>	Specify the tick locations of the vector colorbar
<i>cticklabels</i>	Specify the colorbar ticklabels

Miscellaneous formatoptions

<i>linewidth</i>	Change the linewidth of the arrows
<i>datagrid</i>	Show the grid of the data

Label formatoptions

<i>clabel</i>	Show the colorbar label
---------------	-------------------------

Continued on next page

Table 83 – continued from previous page

<code>clabelprops</code>	Properties of the Colorbar label
<code>clabelsize</code>	Set the size of the Colorbar label
<code>clabelweight</code>	Set the fontweight of the Colorbar label

Post processing formatoptions

<code>post</code>	Apply your own postprocessing script
<code>post_timing</code>	Determine when to run the <code>post</code> 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, vticksize, vtickweight, vticklabels, vticks

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, vtickweight, vtickprops, vticklabels, vticks

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:

label, *labelsize*, *labelweight*

labelsize

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:

label, *labelweight*, *clabelprops*

labelweight

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:

label, *labelsize*, *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, *cticksizes*, *ctickweights*, *ctickprops*, *vcticks*, *vcticksizes*, *vctickweights*, *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

<i>cbar</i>	cbar Formatoption instance in the plotter
<i>cmap</i>	cmap Formatoption instance in the plotter
<i>connections</i>	list() -> new empty list
<i>group</i>	str(object='') -> str
<i>name</i>	str(object='') -> str
<i>priority</i>	int(x=0) -> integer
<i>value2share</i>	The normalization instance

Methods

<i>get_fmt_widget</i> (parent, project)	Open a <code>psy_simple.widget.CMapFmtWidget</code>
<i>update</i> (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

<code>axis_locations</code>	<code>dict()</code> -> new empty dictionary
<code>cbar</code>	cbar <code>Formatoption</code> instance in the plotter
<code>children</code>	<code>list()</code> -> new empty list
<code>data_dependent</code>	<code>bool(x)</code> -> bool
<code>dependencies</code>	<code>list()</code> -> new empty list
<code>group</code>	<code>str(object='')</code> -> str
<code>name</code>	<code>str(object='')</code> -> str
<code>plot</code>	plot <code>Formatoption</code> instance in the plotter

Methods

<code>update(value)</code>	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 *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

<i>bounds</i>	bounds Formatoption instance in the plotter
<i>cbar</i>	cbar Formatoption instance in the plotter
<i>connections</i>	list() -> new empty list
<i>group</i>	str(object='') -> str
<i>name</i>	str(object='') -> str
<i>priority</i>	int(x=0) -> integer

Methods

<code>get_cmap([arr, cmap, N])</code>	Get the <code>matplotlib.colors.Colormap</code> for plotting
<code>get_fmt_widget(parent, project)</code>	Open a <code>psy_simple.widget.CMapFmtWidget</code>
<code>update(value)</code>	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

<code>cbar</code>	cbar Formatoption instance in the plotter
<code>default_formatters</code>	Default locator of the axis of the colorbars
<code>name</code>	<code>str(object='') -> str</code>
<code>plot</code>	plot Formatoption instance in the plotter

Methods

<code>set_default_formatters()</code>	Sets the default formatters that is used for updating to None
<code>set_formatter(formatter)</code>	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`, `cticksiz`, `ctickweight`, `ctickprops`, `vcticks`, `vcticksiz`, `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

<i>cbar</i>	cbar Formatoption instance in the plotter
<i>children</i>	list() -> new empty list
<i>group</i>	str(object='') -> str
<i>name</i>	str(object='') -> str
<i>plot</i>	plot Formatoption instance in the plotter

Methods

<i>update_axis</i> (value)

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

See also:

`cticksize`, `ctickweight`, `cticklabels`, `cticks`, `vticksize`, `vtickweight`, `vticklabels`, `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,    addi-
                                   tional_children=[],    additional_dependencies=[],
                                   **kwargs)
Bases: psy_simple.plotters.CbarOptions, psy_simple.plotters.TickSizeBase
Specify the font size of the colorbar ticklabels
```

Possible types

Attributes

<i>cbar</i>	cbar Formatoption instance in the plotter
<i>ctickprops</i>	ctickprops Formatoption instance in the plotter
<i>dependencies</i>	list() -> new empty list
<i>group</i>	str(object='') -> str
<i>name</i>	str(object='') -> str
<i>plot</i>	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, addi-
                                     tional_children=[], additional_dependencies=[],
                                     **kwargs)

Bases: psy_simple.plotters.CbarOptions, psy_simple.plotters.TickWeightBase

Specify the fontweight of the colorbar ticklabels

```

Possible types

Attributes

<i>cbar</i>	cbar Formatoption instance in the plotter
<i>ctickprops</i>	ctickprops Formatoption instance in the plotter
<i>dependencies</i>	list() -> new empty list
<i>group</i>	str(object='') -> str
<i>name</i>	str(object='') -> str
<i>plot</i>	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

<i>bounds</i>	bounds Formatoption instance in the plotter
<i>cbar</i>	cbar Formatoption instance in the plotter
<i>default_locator</i>	Default locator of the axis of the colorbars
<i>dependencies</i>	list() -> new empty list
<i>name</i>	str(object='') -> str
<i>plot</i>	plot Formatoption instance in the plotter

Methods

<i>set_default_locators</i> (*args, **kwargs)	Sets the default locator that is used for updating to None or int
<i>set_ticks</i> (value)	
<i>update</i> (value)	Method that is call to update the formatoption on the axes
<i>update_axis</i> (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

<i>bounds</i>	bounds Formatoption instance in the plotter
<i>cbarspacing</i>	cbarspacing Formatoption instance in the plotter
<i>cmap</i>	cmap Formatoption instance in the plotter

Continued on next page

Table 99 – continued from previous page

<i>dependencies</i>	<code>list()</code> -> new empty list
<i>extend</i>	extend <code>Formatoption</code> instance in the plotter
<i>figure_positions</i>	<code>set()</code> -> new empty set object
<i>group</i>	<code>str(object='')</code> -> str
<i>init_kwargs</i>	<code>dict</code> key word arguments that are passed to the
<i>levels</i>	levels <code>Formatoption</code> instance in the plotter
<i>name</i>	<code>str(object='')</code> -> str
<i>plot</i>	plot <code>Formatoption</code> instance in the plotter
<i>priority</i>	<code>float(x)</code> -> floating point number
<i>value2share</i>	Those colorbar positions that are directly at the axes

Methods

<i>draw_colorbar(pos)</i>	
<i>finish_update()</i>	Finish the update, initialization and sharing process
<i>initialize_plot(value)</i>	Method that is called when the plot is made the first time
<i>remove([positions])</i>	Method to remove the effects of this formatoption
<i>update(value)</i>	Updates the colorbar
<i>update_colorbar(pos)</i>	

- *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

<i>axis</i>	axis of the colorbar with the ticks. Will be overwritten during
<i>axis_locations</i>	dict() -> new empty dictionary
<i>axisname</i>	
<i>cbar</i>	cbar Formatoption instance in the plotter
<i>children</i>	list() -> new empty list
<i>colorbar</i>	
<i>data</i>	The data that is plotted
<i>dependencies</i>	list() -> new empty list
<i>plot</i>	plot Formatoption instance in the plotter
<i>which</i>	str(object='') -> str

Methods

<i>update</i> (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, addi-
                                     tional_children=[], additional_dependencies=[],
                                     **kwargs)

Bases: psyplot.plotter.Formatoption

Specify the spacing of the bounds in the colorbar

```

Possible types

Attributes

<i>cbar</i>	cbar Formatoption instance in the plotter
<i>connections</i>	list() -> new empty list
<i>group</i>	str(object='') -> str
<i>name</i>	str(object='') -> str

Methods

<i>update</i> (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

<i>arrowsize</i>	Change the size of the arrows
<i>arrowstyle</i>	Change the style of the arrows

Methods

<i>check_data</i> (name, dims, is_unstructured)	A validation method for the data shape
---	--

Color coding formatoptions

<i>color</i>	Set the color for the arrows
<i>vbounds</i>	Specify the boundaries of the vector colorbar
<i>vcbar</i>	Specify the position of the vector plot colorbars
<i>vcbarspacing</i>	Specify the spacing of the bounds in the colorbar
<i>vcmap</i>	Specify the color map
<i>vctickprops</i>	Specify the font properties of the colorbar ticklabels
<i>vcticksize</i>	Specify the font size of the colorbar ticklabels
<i>vctickweight</i>	Specify the fontweight of the colorbar ticklabels
<i>bounds</i>	Specify the boundaries of the colorbar
<i>cbar</i>	Specify the position of the colorbars

Miscellaneous formatoptions

<i>linewidth</i>	Change the linewidth of the arrows
------------------	------------------------------------

Label formatoptions

<i>vclabel</i>	Show the colorbar label of the vector plot
<i>vclabelprops</i>	Properties of the Vector colorbar label
<i>vclabelsize</i>	Set the size of the Vector colorbar label
<i>vclabelweight</i>	Set the fontweight of the Vector colorbar label

Axis tick formatoptions

<i>vcticklabels</i>	Specify the colorbar ticklabels
<i>vcticks</i>	Specify the tick locations of the vector colorbar
<i>cticks</i>	Specify the tick locations of the colorbar

Masking formatoptions

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

Post processing formatoptions

<code>post</code>	Apply your own postprocessing script
<code>post_timing</code>	Determine when to run the <code>post</code> 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. '#eeffff'), 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:

vlabel, *vlabelweight*, *vlabelprops*

vlabelweight

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:

vlabel, *vlabelsize*, *vlabelprops*

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.cmmaps’ key of the `psypplot.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

vticklabels

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

<i>density</i>	Change the density of the arrows
<i>arrowsize</i>	Change the size of the arrows
<i>arrowstyle</i>	Change the style of the arrows

Plot formatoptions

<i>plot</i>	Choose how to visualize a 2-dimensional scalar data field
<i>vplot</i>	Choose the vector plot type

Color coding formatoptions

<i>bounds</i>	Specify the boundaries of the colorbar
<i>cbar</i>	Specify the position of the colorbars
<i>cbarspacing</i>	Specify the spacing of the bounds in the colorbar
<i>cmap</i>	Specify the color map
<i>color</i>	Set the color for the arrows
<i>ctickprops</i>	Specify the font properties of the colorbar ticklabels
<i>cticksize</i>	Specify the font size of the colorbar ticklabels
<i>ctickweight</i>	Specify the fontweight of the colorbar ticklabels
<i>extend</i>	Draw arrows at the side of the colorbar
<i>levels</i>	The levels for the contour plot
<i>miss_color</i>	Set the color for missing values
<i>vbounds</i>	Specify the boundaries of the vector colorbar
<i>vcbar</i>	Specify the position of the vector plot colorbars
<i>vcbarspacing</i>	Specify the spacing of the bounds in the colorbar
<i>vcmap</i>	Specify the color map
<i>vctickprops</i>	Specify the font properties of the colorbar ticklabels
<i>vcticksize</i>	Specify the font size of the colorbar ticklabels
<i>vctickweight</i>	Specify the fontweight of the colorbar ticklabels

Masking formatoptions

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

Label formatoptions

<i>clabel</i>	Show the colorbar label
<i>clabelprops</i>	Properties of the Colorbar label
<i>clabelsize</i>	Set the size of the Colorbar label
<i>clabelweight</i>	Set the fontweight of the Colorbar label
<i>figtitle</i>	Plot a figure title
<i>figtitleprops</i>	Properties of the figure title
<i>figtitlesize</i>	Set the size of the figure title
<i>figtitleweight</i>	Set the fontweight of the figure title
<i>labelprops</i>	Set the font properties of both, x- and y-label
<i>labelsize</i>	Set the size of both, x- and y-label
<i>labelweight</i>	Set the font size of both, x- and y-label
<i>text</i>	Add text anywhere on the plot
<i>title</i>	Show the title
<i>titleprops</i>	Properties of the title
<i>titlesize</i>	Set the size of the title
<i>titleweight</i>	Set the fontweight of the title
<i>vclabel</i>	Show the colorbar label of the vector plot
<i>vclabelprops</i>	Properties of the Vector colorbar label
<i>vclabelsize</i>	Set the size of the Vector colorbar label
<i>vclabelweight</i>	Set the fontweight of the Vector colorbar label
<i>xlabel</i>	Set the x-axis label

Continued on next page

Table 117 – continued from previous page

<i>ylabel</i>	Set the y-axis label
Post processing formatoptions	
<i>post</i>	Apply your own postprocessing script
<i>post_timing</i>	Determine when to run the <i>post</i> formatoption
Axes formatoptions	
<i>axiscolor</i>	Color the x- and y-axes
<i>grid</i>	Display the grid
<i>tight</i>	Automatically adjust the plots.
<i>transpose</i>	Switch x- and y-axes
<i>xlim</i>	Set the x-axis limits
<i>ylim</i>	Set the y-axis limits
Axis tick formatoptions	
<i>cticklabels</i>	Specify the colorbar ticklabels
<i>cticks</i>	Specify the tick locations of the colorbar
<i>vcticklabels</i>	Specify the colorbar ticklabels
<i>vcticks</i>	Specify the tick locations of the vector colorbar
<i>xrotation</i>	Rotate the x-axis ticks
<i>xticklabels</i>	Modify the x-axis ticklabels
<i>xtickprops</i>	Specify the x-axis tick parameters
<i>xticks</i>	Modify the x-axis ticks
<i>yrotation</i>	Rotate the y-axis ticks
<i>yticklabels</i>	Modify the y-axis ticklabels
<i>ytickprops</i>	Specify the y-axis tick parameters
<i>yticks</i>	Modify the y-axis ticks
Miscellaneous formatoptions	
<i>datagrid</i>	Show the grid of the data
<i>interp_bounds</i>	Interpolate grid cell boundaries for 2D plots
<i>linewidth</i>	Change the linewidth of the arrows
<i>sym_lims</i>	Make x- and y-axis symmetric
<i>ticksize</i>	Change the ticksize of the ticklabels
<i>tickweight</i>	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 unstructured 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, vticksize, vtickweight, vticklabels, vticks

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, vtickweight, vtickprops, vticklabels, vticks

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.

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

vcbars

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, vticksize, vtickweight, vticklabels, vticks`

vticksize

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, vtickweight, vtickprops, vticklabels, vticks`

vtickweight

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, vticksize, vtickprops, vticklabels, vticks`

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 `psypplot.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 `psypplot.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` format option.

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 appearance (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 `psypplot.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:

vlabel, vlabelsize, vlabelweight

vlabelsize

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:

vlabel, vlabelweight, vlabelprops

vlabelweight

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:

vlabel, vlabelsize, vlabelprops

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`
 - `sdsc: %(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 happen 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

<i>arrowsize</i>	arrowsize Formatoption instance in the plotter
<i>arrowstyle</i>	arrowstyle Formatoption instance in the plotter
<i>bounds</i>	bounds Formatoption instance in the plotter
<i>cmap</i>	cmap Formatoption instance in the plotter
<i>color</i>	color Formatoption instance in the plotter
<i>data_dependent</i>	bool(x) -> bool
<i>density</i>	density Formatoption instance in the plotter
<i>linewidth</i>	linewidth Formatoption instance in the plotter
<i>transform</i>	transform Formatoption instance in the plotter
<i>transpose</i>	transpose Formatoption instance in the plotter

Methods

<i>update</i> (*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

<i>cbar</i>	cbar Formatoption instance in the plotter
<i>cbounds</i>	cbounds Formatoption instance in the plotter
<i>cmap</i>	cmap Formatoption instance in the plotter
<i>dependencies</i>	list() -> new empty list
<i>name</i>	str(object='') -> str
<i>priority</i>	int(x=0) -> integer

Methods

<i>update</i> (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, addi-
                                tional_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

<i>array</i>	The (masked) data array that is plotted
<i>children</i>	list() -> new empty list

Continued on next page

Table 126 – continued from previous page

<i>name</i>	<code>str(object='') -> str</code>
<i>transform</i>	transform Formatoption instance in the plotter
<i>triangles</i>	The <code>matplotlib.tri.Triangulation</code> instance containing the
<i>xbounds</i>	Boundaries of the x-coordinate
<i>ybounds</i>	Boundaries of the y-coordinate

Methods

<i>remove()</i>	Method to remove the effects of this formatoption
<i>update(value)</i>	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 bandwidth of the kernel size (if the *density* formatoption is set to 'kde')

Possible types**Attributes**

<i>connections</i>	<code>list()</code> -> new empty list
<i>data_dependent</i>	<code>bool(x)</code> -> bool
<i>density</i>	density Formatoption instance in the plotter
<i>dependencies</i>	<code>list()</code> -> new empty list
<i>group</i>	<code>str(object='')</code> -> str
<i>name</i>	<code>str(object='')</code> -> str
<i>priority</i>	<code>int(x=0)</code> -> integer
<i>xrange</i>	xrange Formatoption instance in the plotter
<i>yrange</i>	yrange Formatoption instance in the plotter

Methods

<i>update</i> (<i>value</i>)	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

<code>array</code>	The numpy array of the data
<code>data_dependent</code>	<code>bool(x) -> bool</code>
<code>full_array</code>	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

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

Methods

<code>remove([plot_type])</code>	Method to remove the effects of this formatoption
<code>update(value)</code>	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

<code>allowed_dims</code>	<code>int(x=0) -> integer</code>
<code>allowed_vars</code>	<code>int(x=0) -> integer</code>

Data manipulation formatoptions

<code>bins</code>	Specify the bins of the 2D-Histogramm
<code>coord</code>	Use an alternative variable as x-coordinate

Continued on next page

Table 134 – continued from previous page

<i>density</i>	Specify the method to calculate the density
<i>normed</i>	Specify the normalization of the histogram
<i>precision</i>	Set the precision of the data
<i>xrange</i>	Specify the range of the histogram for the x-dimension
<i>yrange</i>	Specify the range of the histogram for the x-dimension

Color coding formatoptions

<i>bounds</i>	Specify the boundaries of the colorbar
<i>cbar</i>	Specify the position of the colorbars
<i>cbarspacing</i>	Specify the spacing of the bounds in the colorbar
<i>cmap</i>	Specify the color map
<i>ctickprops</i>	Specify the font properties of the colorbar ticklabels
<i>cticksize</i>	Specify the font size of the colorbar ticklabels
<i>ctickweight</i>	Specify the fontweight of the colorbar ticklabels
<i>extend</i>	Draw arrows at the side of the colorbar
<i>levels</i>	The levels for the contour plot
<i>miss_color</i>	Set the color for missing values

Masking formatoptions

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

Label formatoptions

<i>clabel</i>	Show the colorbar label
<i>clabelprops</i>	Properties of the Colorbar label
<i>clabelsize</i>	Set the size of the Colorbar label
<i>clabelweight</i>	Set the fontweight of the Colorbar label
<i>figtitle</i>	Plot a figure title
<i>figtitleprops</i>	Properties of the figure title
<i>figtitlesize</i>	Set the size of the figure title
<i>figtitleweight</i>	Set the fontweight of the figure title
<i>labelprops</i>	Set the font properties of both, x- and y-label
<i>labelsize</i>	Set the size of both, x- and y-label
<i>labelweight</i>	Set the font size of both, x- and y-label
<i>text</i>	Add text anywhere on the plot
<i>title</i>	Show the title
<i>titleprops</i>	Properties of the title
<i>titlesize</i>	Set the size of the title
<i>titleweight</i>	Set the fontweight of the title
<i>xlabel</i>	Set the x-axis label
<i>ylabel</i>	Set the y-axis label

Post processing formatoptions

<i>post</i>	Apply your own postprocessing script
<i>post_timing</i>	Determine when to run the <i>post</i> formatoption

Axes formatoptions

<i>axiscolor</i>	Color the x- and y-axes
<i>grid</i>	Display the grid
<i>tight</i>	Automatically adjust the plots.
<i>transpose</i>	Switch x- and y-axes
<i>xlim</i>	Set the x-axis limits
<i>ylim</i>	Set the y-axis limits

Axis tick formatoptions

<i>cticklabels</i>	Specify the colorbar ticklabels
<i>cticks</i>	Specify the tick locations of the colorbar
<i>xrotation</i>	Rotate the x-axis ticks
<i>xticklabels</i>	Modify the x-axis ticklabels
<i>xtickprops</i>	Specify the x-axis tick parameters
<i>xticks</i>	Modify the x-axis ticks
<i>yrotation</i>	Rotate the y-axis ticks
<i>yticklabels</i>	Modify the y-axis ticklabels
<i>ytickprops</i>	Specify the y-axis tick parameters
<i>yticks</i>	Modify the y-axis ticks

Miscellaneous formatoptions

<i>datagrid</i>	Show the grid of the data
<i>interp_bounds</i>	Interpolate grid cell boundaries for 2D plots
<i>sym_lims</i>	Make x- and y-axis symmetric
<i>ticksize</i>	Change the ticksize of the ticklabels
<i>tickweight</i>	Change the fontweight of the ticks

Plot formatoptions

<i>plot</i>	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 python's *[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 `psypplot.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`, `vtickweight`, `vticklabels`, `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`, `vtickweight`, `vtickprops`, `vticklabels`, `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`, `vtickprops`, `vticklabels`, `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:

label, 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:

label, 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:

label, 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. $0 \leq x, y \leq 1$ 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 appearance (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 `psypplot.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

<i>dtdata</i>	The np.unique data as datetime objects
<i>plot</i>	plot Formatoption instance in the plotter
<i>transpose</i>	transpose Formatoption instance in the plotter

Methods

<i>update(value)</i>	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, addi-  
                                     tional_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

<code>connections</code>	<code>list()</code> -> new empty list
<code>error</code>	error Formatoption instance in the plotter
<code>group</code>	<code>str(object='')</code> -> str
<code>name</code>	<code>str(object='')</code> -> str
<code>priority</code>	<code>int(x=0)</code> -> integer

Methods

<code>update(value)</code>	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=[],      addi-
                                         tional_dependencies=[], **kwargs)
```

Bases: `psyplot.plotter.Formatoption`

Calculation of the error

This formatoption is used to calculate the error range.

Possible types

Attributes

<code>children</code>	<code>list()</code> -> new empty list
<code>data_dependent</code>	<code>bool(x)</code> -> bool
<code>group</code>	<code>str(object='')</code> -> str
<code>mean</code>	mean <code>Formatoption</code> instance in the plotter
<code>name</code>	<code>str(object='')</code> -> str
<code>priority</code>	<code>int(x=0)</code> -> integer
<code>requires_replot</code>	<code>bool(x)</code> -> bool

Methods

<code>update(value)</code>	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

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

Methods

<code>make_plot()</code>	
<code>plot_fill(index, min_range, max_range, c, ...)</code>	
<code>remove()</code>	Method to remove the effects of this formatoption
<code>update(value)</code>	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

<code>connections</code>	list() -> new empty list
<code>group</code>	str(object='') -> str
<code>name</code>	str(object='') -> str
<code>plot</code>	plot Formatoption instance in the plotter

Methods

<code>update(value)</code>	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

<code>allowed_dims</code>	<code>int(x=0) -> integer</code>
---------------------------	-------------------------------------

Data manipulation formatoptions

<code>coord</code>	Use an alternative variable as x-coordinate
<code>err_calc</code>	Calculation of the error
<code>mean</code>	Determine how the error is visualized

Masking formatoptions

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

Color coding formatoptions

<code>color</code>	Set the color coding
<code>erroralpha</code>	Set the alpha value for the error range

Label formatoptions

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

Post processing formatoptions

<i>post</i>	Apply your own postprocessing script
<i>post_timing</i>	Determine when to run the <i>post</i> formatoption

Axes formatoptions

<i>axiscolor</i>	Color the x- and y-axes
<i>grid</i>	Display the grid
<i>tight</i>	Automatically adjust the plots.
<i>transpose</i>	Switch x- and y-axes
<i>xlim</i>	Set the x-axis limits
<i>ylim</i>	Set the y-axis limits

Axis tick formatoptions

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

Miscellaneous formatoptions

<i>legend</i>	Draw a legend
<i>legendlabels</i>	Set the labels of the arrays in the legend
<i>linewidth</i>	Choose the width of the lines
<i>marker</i>	Choose the marker for points
<i>markersize</i>	Choose the size of the markers for points
<i>sym_lims</i>	Make x- and y-axis symmetric
<i>ticksize</i>	Change the ticksize of the ticklabels

Continued on next page

Table 161 – continued from previous page

<i>tickweight</i>	Change the fontweight of the ticks
Plot formatoptions	
<i>error</i>	Visualize the error range
<i>plot</i>	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* format option.

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. $0 \leq x, y \leq 1$ 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 appearance (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 happen 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 `psypplot.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 matplotlib's rcParams
- *float* – The width of the lines

marker

Choose the marker for points

Possible types

- *None* – Use the default from matplotlib's 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 matplotlib's 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

<code>group</code>	<code>str(object='') -> str</code>
<code>name</code>	<code>str(object='') -> str</code>

Methods

<code>update(value)</code>	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

<code>array</code>	The numpy array of the data
<code>coord</code>	coord Formatoption instance in the plotter
<code>data_dependent</code>	bool(x) -> bool
<code>dependencies</code>	list() -> new empty list
<code>group</code>	str(object='') -> str
<code>name</code>	str(object='') -> str
<code>plot</code>	plot Formatoption instance in the plotter
<code>priority</code>	int(x=0) -> integer
<code>transpose</code>	transpose Formatoption instance in the plotter

Methods

<code>set_limit(*args)</code>	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, addi-
                                tional_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

<code>data_dependent</code>	<code>bool(x) -> bool</code>
<code>dependencies</code>	<code>list() -> new empty list</code>
<code>group</code>	<code>str(object='') -> str</code>
<code>name</code>	<code>str(object='') -> str</code>
<code>precision</code>	precision Formatoption instance in the plotter
<code>priority</code>	<code>int(x=0) -> integer</code>

Methods

<code>update(value)</code>	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

<i>update</i> (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

<i>children</i>	list() -> new empty list
<i>xlabel</i>	xlabel Formatoption instance in the plotter
<i>ylabel</i>	ylabel Formatoption instance in the plotter

Methods

<i>update</i> (<i>value</i>)	Method that is call to update the formatoption on the axes
<i>update_axis</i> (<i>value</i>)	

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, addi-
                                     tional_children=[], additional_dependencies=[],
                                     **kwargs)
```

Bases: *psy_simple.plotters.LabelOptions*

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

Possible types

Attributes

<i>children</i>	list() -> new empty list
<i>group</i>	str(object='') -> str
<i>labelsize</i>	labelsize Formatoption instance in the plotter
<i>labelweight</i>	labelweight Formatoption instance in the plotter
<i>name</i>	str(object='') -> str
<i>xlabel</i>	xlabel Formatoption instance in the plotter
<i>ylabel</i>	ylabel Formatoption instance in the plotter

Methods

<i>update_axis</i> (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, labelsizes, labelweight

Parameters

- **key** (*str*) – formatoption key in the *plotter*
- **plotter** (*psypplot.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 *psypplot.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', 'labelsizes', 'labelweight']
```

```
group = 'labels'
```

labelsizes

labelsizes 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

<i>group</i>	str(object='') -> str
<i>labelprops</i>	labelprops Formatoption instance in the plotter
<i>name</i>	str(object='') -> str
<i>parents</i>	list() -> new empty list

Continued on next page

Table 175 – continued from previous page

<code>xlabel</code>	xlabel Formatoption instance in the plotter
<code>ylabel</code>	ylabel Formatoption instance in the plotter

Methods

<code>update_axis(value)</code>

- *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

<code>group</code>	<code>str(object='') -> str</code>
<code>labelprops</code>	labelprops Formatoption instance in the plotter
<code>name</code>	<code>str(object='') -> str</code>
<code>parents</code>	<code>list()</code> -> new empty list
<code>xlabel</code>	xlabel Formatoption instance in the plotter
<code>ylabel</code>	ylabel Formatoption instance in the plotter

Methods

<code>update_axis(value)</code>

- *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** (*psypplot.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 `psypplot.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**

<i>color</i>	color Formatoption instance in the plotter
<i>dependencies</i>	list() -> new empty list
<i>legendlabels</i>	legendlabels Formatoption instance in the plotter
<i>marker</i>	marker Formatoption instance in the plotter
<i>name</i>	str(object='') -> str
<i>plot</i>	plot Formatoption instance in the plotter

Methods

<i>get_artists_and_labels()</i>	
<i>remove()</i>	Method to remove the effects of this formatoption
<i>update(value)</i>	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

<code>data_dependent</code>	<code>bool(x) -> bool</code>
<code>name</code>	<code>str(object='') -> str</code>

Methods

<code>update(value)</code>	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

<code>children</code>	<code>list()</code> -> new empty list
<code>connections</code>	<code>list()</code> -> new empty list
<code>group</code>	<code>str(object='')</code> -> str
<code>plot</code>	plot Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>value2share</code>	The value that is passed to shared formatoptions (by default, the

Methods

<code>set_limit(min_val, max_val)</code>	The method to set the minimum and maximum limit
<code>update(value)</code>	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

<i>group</i>	<code>str(object='') -> str</code>
<i>name</i>	<code>str(object='') -> str</code>
<i>priority</i>	<code>int(x=0) -> integer</code>
<i>value2pickle</i>	The value that can be used when pickling the information of the project
<i>value2share</i>	The value that is passed to shared formatoptions (by default, the

Methods

<i>update</i> (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.cmmaps’ 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

<code>children</code>	<code>list()</code> -> new empty list
<code>color</code>	color <code>Formatoption</code> instance in the plotter
<code>data_dependent</code>	<code>bool(x)</code> -> bool
<code>group</code>	<code>str(object='')</code> -> str
<code>marker</code>	marker <code>Formatoption</code> instance in the plotter
<code>name</code>	<code>str(object='')</code> -> str
<code>plot_fmt</code>	<code>bool(x)</code> -> bool
<code>priority</code>	<code>float(x)</code> -> floating point number
<code>transpose</code>	transpose <code>Formatoption</code> instance in the plotter

Methods

<code>make_plot()</code>	
<code>plot_arr(arr, c, ls, m)</code>	
<code>remove()</code>	Method to remove the effects of this formatoption
<code>update(value)</code>	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

<i>allowed_vars</i>	The number variables that one data array visualized by this plotter might have.
---------------------	---

Data manipulation formatoptions

<i>coord</i>	Use an alternative variable as x-coordinate
--------------	---

Plot formatoptions

<i>error</i>	Visualize the error range
<i>plot</i>	Choose the line style of the plot

Color coding formatoptions

<i>erroralpha</i>	Set the alpha value for the error range
<i>color</i>	Set the color coding

Miscellaneous formatoptions

<i>linewidth</i>	Choose the width of the lines
<i>marker</i>	Choose the marker for points
<i>markersize</i>	Choose the size of the markers for points
<i>legend</i>	Draw a legend
<i>legendlabels</i>	Set the labels of the arrays in the legend
<i>sym_lims</i>	Make x- and y-axis symmetric
<i>ticksize</i>	Change the ticksize of the ticklabels
<i>tickweight</i>	Change the fontweight of the ticks

Masking formatoptions

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

Label formatoptions

<i>figtitle</i>	Plot a figure title
<i>figtitleprops</i>	Properties of the figure title
<i>figtitlesize</i>	Set the size of the figure title
<i>figtitleweight</i>	Set the fontweight of the figure title
<i>labelprops</i>	Set the font properties of both, x- and y-label
<i>labelsize</i>	Set the size of both, x- and y-label
<i>labelweight</i>	Set the font size of both, x- and y-label
<i>text</i>	Add text anywhere on the plot
<i>title</i>	Show the title
<i>titleprops</i>	Properties of the title

Continued on next page

Table 195 – continued from previous page

<code>titlesize</code>	Set the size of the title
<code>titleweight</code>	Set the fontweight of the title
<code>xlabel</code>	Set the x-axis label
<code>ylabel</code>	Set the y-axis label

Post processing formatoptions

<code>post</code>	Apply your own postprocessing script
<code>post_timing</code>	Determine when to run the <code>post</code> formatoption

Axes formatoptions

<code>axiscolor</code>	Color the x- and y-axes
<code>grid</code>	Display the grid
<code>tight</code>	Automatically adjust the plots.
<code>transpose</code>	Switch x- and y-axes
<code>xlim</code>	Set the x-axis limits
<code>ylim</code>	Set the y-axis limits

Axis tick formatoptions

<code>xrotation</code>	Rotate the x-axis ticks
<code>xticklabels</code>	Modify the x-axis ticklabels
<code>xtickprops</code>	Specify the x-axis tick parameters
<code>xticks</code>	Modify the x-axis ticks
<code>yrotation</code>	Rotate the y-axis ticks
<code>yticklabels</code>	Modify the y-axis ticklabels
<code>ytickprops</code>	Specify the y-axis tick parameters
<code>yticks</code>	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 variables where 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` format option.

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 `psypplot.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 appearance (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 `psypplot.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* format option.

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 happen 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 `psypplot.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**

<i>connections</i>	list() -> new empty list
<i>plot</i>	plot Formatoption instance in the plotter
<i>priority</i>	int(x=0) -> integer

Methods

<i>update</i> (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

<i>priority</i>	<code>int(x=0)</code> -> integer
-----------------	----------------------------------

Methods

<i>update</i> (<i>value</i>)	Method that is call to update the formatoption on the axes
--------------------------------	--

- *None* – Use the default from matplotlib's 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

<i>connections</i>	list() -> new empty list
<i>plot</i>	plot Formatoption instance in the plotter
<i>priority</i>	int(x=0) -> integer

Methods

<i>update</i> (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

<i>data_dependent</i>	<code>bool(x) -> bool</code>
<i>group</i>	<code>str(object='') -> str</code>
<i>name</i>	<code>str(object='') -> str</code>
<i>priority</i>	<code>int(x=0) -> integer</code>
<i>requires_replot</i>	<code>bool(x) -> bool</code>

Methods

<i>update</i> (<i>value</i>)	Method that is call to update the <i>formatoption</i> 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, addi-  
                                     tional_children=[], additional_dependencies=[],  
                                     **kwargs)
```

Bases: `psyplot.plotter.Formatoption`

Set the color for missing values

Possible types

Attributes

<code>connections</code>	list() -> new empty list
<code>dependencies</code>	list() -> new empty list
<code>group</code>	str(object='') -> str
<code>name</code>	str(object='') -> str
<code>plot</code>	plot Formatoption instance in the plotter
<code>priority</code>	int(x=0) -> integer
<code>transform</code>	transform Formatoption instance in the plotter
<code>triangles</code>	The <code>matplotlib.tri.Triangulation</code> instance containing the
<code>update_after_plot</code>	bool(x) -> bool

Methods

<code>remove()</code>	Method to remove the effects of this formatoption
<code>update(value)</code>	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 `matplotliblib.axes.Axes.clear()` method.

```
transform
    transform Formatoption instance in the plotter
```

```
triangles
    The matplotliblib.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, addi-
                                         tional_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

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

Methods

<code>hist2d(da, **kwargs)</code>	Make the two dimensional histogram
<code>update(value)</code>	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

<code>add2format_coord(x, y)</code>	Additional information for the <code>format_coord()</code>
<code>get_xyz_1d(xcoord, x, ycoord, y, data)</code>	Get closest x, y and z for the given <i>x</i> and <i>y</i> in <i>data</i> for
<code>get_xyz_2d(xcoord, x, ycoord, y, data)</code>	Get closest x, y and z for the given <i>x</i> and <i>y</i> in <i>data</i> for
<code>get_xyz_tri(xcoord, x, ycoord, y, data)</code>	Get closest x, y and z for the given <i>x</i> and <i>y</i> in <i>data</i> for
<code>make_plot()</code>	
<code>remove()</code>	Method to remove the effects of this formatoption
<code>update(value)</code>	Method that is call to update the formatoption on the axes

Attributes

<code>array</code>	The (masked) data array that is plotted
<code>bounds</code>	bounds Formatoption instance in the plotter
<code>children</code>	list() -> new empty list
<code>cmap</code>	cmap Formatoption instance in the plotter
<code>data_dependent</code>	bool(x) -> bool
<code>dependencies</code>	list() -> new empty list
<code>format_coord</code>	The function that can replace the axes.format_coord method
<code>group</code>	str(object='') -> str
<code>interp_bounds</code>	interp_bounds Formatoption instance in the plotter
<code>levels</code>	levels Formatoption instance in the plotter
<code>mappable</code>	Returns the mappable that can be used for colorbars
<code>name</code>	str(object='') -> str
<code>plot_fmt</code>	bool(x) -> bool
<code>priority</code>	int(x=0) -> integer
<code>triangles</code>	The <code>matplotlib.tri.Triangulation</code> instance containing the
<code>xbounds</code>	Boundaries of the x-coordinate
<code>xcoord</code>	The x coordinate <code>xarray.Variable</code>
<code>ybounds</code>	Boundaries of the y-coordinate
<code>ycoord</code>	The y coordinate <code>xarray.Variable</code>

- *None* – Don't make any plotting
- *'mesh'* – Use the `matplotlib.pyplot.pcolormesh()` function to make the plot or the `matplotlib.pyplot.tripcolor()` for an unstructured 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

<code>bins</code>	bins <code>Formatoption</code> instance in the plotter
<code>coord</code>	coord <code>Formatoption</code> instance in the plotter
<code>data_dependent</code>	<code>bool(x) -> bool</code>
<code>dependencies</code>	<code>list() -> new empty list</code>
<code>group</code>	<code>str(object='') -> str</code>
<code>name</code>	<code>str(object='') -> str</code>
<code>normed</code>	normed <code>Formatoption</code> instance in the plotter
<code>precision</code>	precision <code>Formatoption</code> instance in the plotter
<code>priority</code>	<code>int(x=0) -> integer</code>
<code>xrange</code>	xrange <code>Formatoption</code> instance in the plotter
<code>yrange</code>	yrange <code>Formatoption</code> instance in the plotter

Methods

<code>update(value)</code>	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 python's [\[statsmodels\]](#) 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,    en-
                                              able_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

<i>bounds</i>	Specify the boundaries of the colorbar
<i>cbar</i>	Specify the position of the colorbars

Axis tick formatoptions

<i>cticks</i>	Specify the tick locations of the colorbar
---------------	--

Masking formatoptions

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

Post processing formatoptions

<i>post</i>	Apply your own postprocessing script
<i>post_timing</i>	Determine when to run the <i>post</i> 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

<i>allowed_dims</i>	The number of allowed dimensions in the for the visualization.
---------------------	--

Methods

<i>check_data</i> (name, dims, is_unstructured)	A validation method for the data shape
---	--

Color coding formatoptions

<i>miss_color</i>	Set the color for missing values
<i>bounds</i>	Specify the boundaries of the colorbar
<i>cbar</i>	Specify the position of the colorbars
<i>cbarspacing</i>	Specify the spacing of the bounds in the colorbar
<i>cmap</i>	Specify the color map
<i>ctickprops</i>	Specify the font properties of the colorbar ticklabels
<i>cticksize</i>	Specify the font size of the colorbar ticklabels
<i>ctickweight</i>	Specify the fontweight of the colorbar ticklabels
<i>extend</i>	Draw arrows at the side of the colorbar

Label formatoptions

<i>clabel</i>	Show the colorbar label
<i>clabelprops</i>	Properties of the Colorbar label

Continued on next page

Table 222 – continued from previous page

<code>clabelsize</code>	Set the size of the Colorbar label
<code>clabelweight</code>	Set the fontweight of the Colorbar label

Axis tick formatoptions

<code>cticklabels</code>	Specify the colorbar ticklabels
<code>cticks</code>	Specify the tick locations of the colorbar

Post processing formatoptions

<code>post</code>	Apply your own postprocessing script
<code>post_timing</code>	Determine when to run the <code>post</code> formatoption

Miscellaneous formatoptions

<code>datagrid</code>	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`, `vticksize`, `vtickweight`, `vticklabels`, `vticks`

cticksiz

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:

`cticksiz`, `ctickprops`, `cticklabels`, `cticks`, `vcticksiz`, `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:

label, 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:

label, 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:

label, 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`, `cticksizes`, `ctickweights`, `ctickprops`, `vcticks`, `vcticksizes`, `vctickweights`, `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`

Miscellaneous formatoptions

<code>interp_bounds</code>	Interpolate grid cell boundaries for 2D plots
<code>datagrid</code>	Show the grid of the data
<code>sym_lims</code>	Make x- and y-axis symmetric
<code>ticksize</code>	Change the ticksize of the ticklabels
<code>tickweight</code>	Change the fontweight of the ticks

Color coding formatoptions

<code>levels</code>	The levels for the contour plot
<code>bounds</code>	Specify the boundaries of the colorbar
<code>cbar</code>	Specify the position of the colorbars
<code>cbarspacing</code>	Specify the spacing of the bounds in the colorbar
<code>cmap</code>	Specify the color map
<code>ctickprops</code>	Specify the font properties of the colorbar ticklabels
<code>cticksize</code>	Specify the font size of the colorbar ticklabels
<code>ctickweight</code>	Specify the fontweight of the colorbar ticklabels
<code>extend</code>	Draw arrows at the side of the colorbar
<code>miss_color</code>	Set the color for missing values

Plot formatoptions

<code>plot</code>	Specify the plotting method
-------------------	-----------------------------

Axes formatoptions

<i>transpose</i>	Switch x- and y-axes
<i>xlim</i>	Set the x-axis limits
<i>ylim</i>	Set the y-axis limits
<i>axiscolor</i>	Color the x- and y-axes
<i>grid</i>	Display the grid
<i>tight</i>	Automatically adjust the plots.

Axis tick formatoptions

<i>xticks</i>	Modify the x-axis ticks
<i>yticks</i>	Modify the y-axis ticks
<i>cticklabels</i>	Specify the colorbar ticklabels
<i>cticks</i>	Specify the tick locations of the colorbar
<i>xrotation</i>	Rotate the x-axis ticks
<i>xticklabels</i>	Modify the x-axis ticklabels
<i>xtickprops</i>	Specify the x-axis tick parameters
<i>yrotation</i>	Rotate the y-axis ticks
<i>yticklabels</i>	Modify the y-axis ticklabels
<i>ytickprops</i>	Specify the y-axis tick parameters

Masking formatoptions

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

Label formatoptions

<i>clabel</i>	Show the colorbar label
<i>clabelprops</i>	Properties of the Colorbar label
<i>clabelsize</i>	Set the size of the Colorbar label
<i>clabelweight</i>	Set the fontweight of the Colorbar label
<i>figtitle</i>	Plot a figure title
<i>figtitleprops</i>	Properties of the figure title
<i>figtitlesize</i>	Set the size of the figure title
<i>figtitleweight</i>	Set the fontweight of the figure title
<i>labelprops</i>	Set the font properties of both, x- and y-label
<i>labelsize</i>	Set the size of both, x- and y-label
<i>labelweight</i>	Set the font size of both, x- and y-label
<i>text</i>	Add text anywhere on the plot
<i>title</i>	Show the title
<i>titleprops</i>	Properties of the title
<i>titlesize</i>	Set the size of the title
<i>titleweight</i>	Set the fontweight of the title
<i>xlabel</i>	Set the x-axis label
<i>ylabel</i>	Set the y-axis label

Post processing formatoptions

<code>post</code>	Apply your own postprocessing script
<code>post_timing</code>	Determine when to run the <code>post</code> 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`, `vticksize`, `vtickweight`, `vticklabels`, `vticks`

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:

label, 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:

label, 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, labelsizes, labelweight

labelsizes

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. $0 \leq x, y \leq 1$ 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 appearance (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`
 - `sdsc: %(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 happen 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`, `cticksizes`, `ctickweights`, `ctickprops`, `vcticks`, `vcticksizes`, `vtickweights`, `vtickprops`

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**

<code>array</code>	The (masked) data array that is plotted
<code>bounds</code>	bounds Formatoption instance in the plotter
<code>cmap</code>	cmap Formatoption instance in the plotter
<code>data_dependent</code>	<code>bool(x) -> bool</code>
<code>dependencies</code>	<code>list() -> new empty list</code>
<code>interp_bounds</code>	interp_bounds Formatoption instance in the plotter
<code>levels</code>	levels Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>triangles</code>	The <code>matplotlib.tri.Triangulation</code> instance containing the
<code>xbounds</code>	Boundaries of the x-coordinate
<code>xcoord</code>	The x coordinate <code>xarray.Variable</code>
<code>ybounds</code>	Boundaries of the y-coordinate
<code>ycoord</code>	The y coordinate <code>xarray.Variable</code>

- *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

<i>allowed_dims</i>	The number of allowed dimensions in the for the visualization.
<i>allowed_vars</i>	The number variables that one data array visualized by this plotter might have.

Axes formatoptions

<i>axiscolor</i>	Color the x- and y-axes
<i>grid</i>	Display the grid
<i>transpose</i>	Switch x- and y-axes
<i>xlim</i>	Set the x-axis limits
<i>ylim</i>	Set the y-axis limits
<i>tight</i>	Automatically adjust the plots.

Methods

<i>check_data</i> (name, dims[, is_unstructured])	A validation method for the data shape
---	--

Color coding formatoptions

<i>color</i>	Set the color coding
--------------	----------------------

Miscellaneous formatoptions

<i>legend</i>	Draw a legend
<i>legendlabels</i>	Set the labels of the arrays in the legend
<i>sym_lims</i>	Make x- and y-axis symmetric
<i>ticksize</i>	Change the ticksize of the ticklabels
<i>tickweight</i>	Change the fontweight of the ticks

Masking formatoptions

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

Label formatoptions

<i>figtitle</i>	Plot a figure title
<i>figtitleprops</i>	Properties of the figure title
<i>figtitlesize</i>	Set the size of the figure title
<i>figtitleweight</i>	Set the fontweight of the figure title
<i>labelprops</i>	Set the font properties of both, x- and y-label
<i>labelsize</i>	Set the size of both, x- and y-label
<i>labelweight</i>	Set the font size of both, x- and y-label

Continued on next page

Table 241 – continued from previous page

<i>text</i>	Add text anywhere on the plot
<i>title</i>	Show the title
<i>titleprops</i>	Properties of the title
<i>titlesize</i>	Set the size of the title
<i>titleweight</i>	Set the fontweight of the title
<i>xlabel</i>	Set the x-axis label
<i>ylabel</i>	Set the y-axis label

Post processing formatoptions

<i>post</i>	Apply your own postprocessing script
<i>post_timing</i>	Determine when to run the <i>post</i> formatoption

Axis tick formatoptions

<i>xrotation</i>	Rotate the x-axis ticks
<i>xticklabels</i>	Modify the x-axis ticklabels
<i>xtickprops</i>	Specify the x-axis tick parameters
<i>xticks</i>	Modify the x-axis ticks
<i>yrotation</i>	Rotate the y-axis ticks
<i>yticklabels</i>	Modify the y-axis ticklabels
<i>ytickprops</i>	Specify the y-axis tick parameters
<i>yticks</i>	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 `psypplot.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 `psypplot.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 appearance (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 `psypplot.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**

<i>arrowsize</i>	arrowsize Formatoption instance in the plotter
<i>arrowstyle</i>	arrowstyle Formatoption instance in the plotter
<i>bounds</i>	bounds Formatoption instance in the plotter
<i>cmap</i>	cmap Formatoption instance in the plotter
<i>color</i>	color Formatoption instance in the plotter
<i>data_dependent</i>	bool(x) -> bool
<i>density</i>	density Formatoption instance in the plotter
<i>linewidth</i>	linewidth Formatoption instance in the plotter
<i>transform</i>	transform Formatoption instance in the plotter
<i>transpose</i>	transpose Formatoption instance in the plotter

Methods

<i>set_value</i> (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:

psypplot.plotter.maps.VectorPlotter

Plot formatoptions

<i>plot</i>	Choose the vector plot type
-------------	-----------------------------

Axes formatoptions	
<i>xlim</i>	Set the x-axis limits
<i>ylim</i>	Set the y-axis limits
<i>axiscolor</i>	Color the x- and y-axes
<i>grid</i>	Display the grid
<i>tight</i>	Automatically adjust the plots.
<i>transpose</i>	Switch x- and y-axes

Axis tick formatoptions

<i>xticks</i>	Modify the x-axis ticks
<i>yticks</i>	Modify the y-axis ticks
<i>cticklabels</i>	Specify the colorbar ticklabels
<i>cticks</i>	Specify the tick locations of the vector colorbar
<i>xrotation</i>	Rotate the x-axis ticks
<i>xticklabels</i>	Modify the x-axis ticklabels
<i>xtickprops</i>	Specify the x-axis tick parameters
<i>yrotation</i>	Rotate the y-axis ticks
<i>yticklabels</i>	Modify the y-axis ticklabels
<i>ytickprops</i>	Specify the y-axis tick parameters

Color coding formatoptions

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

Masking formatoptions

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

Label formatoptions

<i>clabel</i>	Show the colorbar label
<i>clabelprops</i>	Properties of the Colorbar label
<i>clabelsize</i>	Set the size of the Colorbar label
<i>clabelweight</i>	Set the fontweight of the Colorbar label
<i>figtitle</i>	Plot a figure title
<i>figtitleprops</i>	Properties of the figure title
<i>figtitlesize</i>	Set the size of the figure title
<i>figtitleweight</i>	Set the fontweight of the figure title
<i>labelprops</i>	Set the font properties of both, x- and y-label
<i>labelsize</i>	Set the size of both, x- and y-label
<i>labelweight</i>	Set the font size of both, x- and y-label
<i>text</i>	Add text anywhere on the plot
<i>title</i>	Show the title
<i>titleprops</i>	Properties of the title
<i>titlesize</i>	Set the size of the title

Continued on next page

Table 251 – continued from previous page

<i>titleweight</i>	Set the fontweight of the title
<i>xlabel</i>	Set the x-axis label
<i>ylabel</i>	Set the y-axis label

Post processing formatoptions

<i>post</i>	Apply your own postprocessing script
<i>post_timing</i>	Determine when to run the <i>post</i> formatoption

Vector plot formatoptions

<i>arrowsize</i>	Change the size of the arrows
<i>arrowstyle</i>	Change the style of the arrows
<i>density</i>	Change the density of the arrows

Miscellaneous formatoptions

<i>datagrid</i>	Show the grid of the data
<i>linewidth</i>	Change the linewidth of the arrows
<i>sym_lims</i>	Make x- and y-axis symmetric
<i>ticksize</i>	Change the ticksize of the ticklabels
<i>tickweight</i>	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 `psysplot.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, vticksize, vtickweight, vticklabels, vticks

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, vtickweight, vtickprops, vticklabels, vticks

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:

label, labelsize, labelweight

labelsize

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:

label, labelweight, labelprops

labelweight

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:

label, labelsize, labelprops

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 `rc-Params['texts.delimiter']` item is used.
- This is the title of the whole figure! For the title of this specific subplot, see the *title* format option.

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 appearance (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* format option.

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 `psypplot.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
<i>'b'</i>	blue
<i>'g'</i>	green
<i>'r'</i>	red
<i>'c'</i>	cyan
<i>'m'</i>	magenta
<i>'y'</i>	yellow
<i>'k'</i>	black
<i>'w'</i>	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, *cticksizes*, *ctickweights*, *ctickprops*, *vcticks*, *vcticksizes*, *vctickweights*, *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=[],      addi-
                                          tional_dependencies=[], **kwargs)
```

Bases: `psyplot.plotter.Formatoption`

Make x- and y-axis symmetric

Possible types

Attributes

<i>dependencies</i>	list() -> new empty list
<i>name</i>	str(object='') -> str
<i>xlim</i>	xlim Formatoption instance in the plotter
<i>ylim</i>	ylim Formatoption instance in the plotter

Methods

<i>update</i> (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

<i>set_default_formatters</i> ([which])	Sets the default formatters that is used for updating to None
<i>set_formatter</i> (formatter[, which])	Sets a given formatter
<i>update</i> (value)	Method that is call to update the formatoption on the axes

Attributes

<i>transpose</i>	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

<i>axis</i>	The axis on the axes to modify the ticks of
<i>dependencies</i>	list() -> new empty list
<i>group</i>	str(object='') -> str
<i>transpose</i>	transpose Formatoption instance in the plotter

Methods

<i>initialize_plot</i> (value)	Method that is called when the plot is made the first time
<i>set_default_formatters</i> ()	Sets the default formatters that is used for updating to None
<i>set_formatter</i> (formatter)	Sets a given formatter
<i>set_stringformatter</i> (s)	

Continued on next page

Table 260 – continued from previous page

<code>set_ticklabels(labels)</code>	Sets the given tick labels
<code>update_axis(value)</code>	

- *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**

<code>axisname</code>	The name of the axis (either 'x' or 'y')
-----------------------	--

Methods

<code>update_axis(value)</code>

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

<i>dependencies</i>	list() -> new empty list
<i>name</i>	str(object='') -> str
<i>xtickprops</i>	xtickprops Formatoption instance in the plotter
<i>ytickprops</i>	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

<code>dependencies</code>	<code>list()</code> -> new empty list
<code>name</code>	<code>str(object='')</code> -> str
<code>xtickprops</code>	xtickprops Formatoption instance in the plotter
<code>ytickprops</code>	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

<code>axis</code>	
<code>dependencies</code>	list() -> new empty list
<code>group</code>	str(object='') -> str
<code>plot</code>	plot Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter

Methods

<code>get_locator()</code>	
<code>initialize_plot(value)</code>	Method that is called when the plot is made the first time
<code>set_default_locators([which])</code>	Sets the default locator that is used for updating to None or int

Continued on next page

Table 268 – continued from previous page

<code>set_locator(locator)</code>	Sets the locator corresponding of the axis
<code>set_ticks(value)</code>	
<code>update_axis(value)</code>	

- *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

<code>group</code>	<code>str(object='') -> str</code>
--------------------	---------------------------------------

Methods

<code>update(value)</code>	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

<code>get_x(arr)</code>	
<code>get_y(arr)</code>	
<code>initialize_plot(value)</code>	Method that is called when the plot is made the first time
<code>update(value)</code>	Method that is call to update the formatoption on the axes

Attributes

<code>group</code>	<code>str(object='') -> str</code>
<code>name</code>	<code>str(object='') -> str</code>
<code>priority</code>	<code>int(x=0) -> integer</code>

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, addi-
                                tional_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

<code>cbar</code>	cbar Formatoption instance in the plotter
<code>plot</code>	plot Formatoption instance in the plotter

Possible types

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

See also:

`vlabelsize`, `vlabelweight`, `vlabelprops`

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

<code>array</code>	The numpy array of the data
<code>cbar</code>	cbar Formatoption instance in the plotter
<code>cmap</code>	cmap Formatoption instance in the plotter

Continued on next page

Table 276 – continued from previous page

<code>color</code>	color Formatoption instance in the plotter
<code>parents</code>	list() -> new empty list

Methods

<code>update(*args, **kwargs)</code>	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**

<code>data_dependent</code>	<code>bool(x) -> bool</code>
<code>dependencies</code>	<code>list() -> new empty list</code>
<code>plot</code>	plot Formatoption instance in the plotter
<code>priority</code>	<code>int(x=0) -> integer</code>
<code>transpose</code>	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

<i>bounds</i>	bounds Formatoption instance in the plotter
<i>cbarspacing</i>	cbarspacing Formatoption instance in the plotter
<i>cmap</i>	cmap Formatoption instance in the plotter
<i>color</i>	color Formatoption instance in the plotter
<i>dependencies</i>	list() -> new empty list
<i>extend</i>	extend Formatoption instance in the plotter
<i>levels</i>	levels Formatoption instance in the plotter
<i>plot</i>	plot Formatoption instance in the plotter
<i>priority</i>	int(x=0) -> integer

Methods

<i>update</i> (*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**

<i>bounds</i>	bounds Formatoption instance in the plotter
<i>cmap</i>	cmap Formatoption instance in the plotter
<i>dependencies</i>	list() -> new empty list
<i>group</i>	str(object='') -> str
<i>name</i>	str(object='') -> str
<i>plot</i>	plot Formatoption instance in the plotter
<i>transpose</i>	transpose Formatoption instance in the plotter

Methods

<i>update</i> (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**

<code>name</code>	<code>str(object='') -> str</code>
<code>plot</code>	plot Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter

Methods

<code>update(value)</code>	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**

<code>add2format_coord(x, y)</code>	Additional information for the <code>format_coord()</code>
<code>get_xyz_1d(xcoord, x, ycoord, y, u, v)</code>	Get closest x, y and z for the given <i>x</i> and <i>y</i> in <i>data</i> for
<code>get_xyz_2d(xcoord, x, ycoord, y, u, v)</code>	Get closest x, y and z for the given <i>x</i> and <i>y</i> in <i>data</i> for
<code>get_xyz_tri(xcoord, x, ycoord, y, u, v)</code>	Get closest x, y and z for the given <i>x</i> and <i>y</i> in <i>data</i> for
<code>make_plot()</code>	
<code>remove()</code>	Method to remove the effects of this formatoption
<code>update(value)</code>	Method that is call to update the formatoption on the axes

Attributes

<code>arrowsize</code>	arrowsize Formatoption instance in the plotter
<code>arrowstyle</code>	arrowstyle Formatoption instance in the plotter
<code>bounds</code>	bounds Formatoption instance in the plotter
<code>children</code>	list() -> new empty list
<code>cmap</code>	cmap Formatoption instance in the plotter
<code>color</code>	color Formatoption instance in the plotter

Continued on next page

Table 287 – continued from previous page

<i>connections</i>	list() -> new empty list
<i>data_dependent</i>	bool(x) -> bool
<i>density</i>	density Formatoption instance in the plotter
<i>format_coord</i>	The function that can replace the axes.format_coord method
<i>group</i>	str(object='') -> str
<i>linewidth</i>	linewidth Formatoption instance in the plotter
<i>mappable</i>	The mappable, i.e.
<i>name</i>	str(object='') -> str
<i>plot_fmt</i>	bool(x) -> bool
<i>priority</i>	int(x=0) -> integer
<i>transform</i>	transform Formatoption instance in the plotter
<i>transpose</i>	transpose Formatoption instance in the plotter
<i>xcoord</i>	The x coordinate <code>xarray.Variable</code>
<i>ycoord</i>	The y coordinate <code>xarray.Variable</code>

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', 'linewidth']

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

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

Methods

<code>make_plot()</code>	
<code>remove()</code>	Method to remove the effects of this formatoption
<code>update(value)</code>	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

<i>plot</i>	Choose how to make the violin plot
-------------	------------------------------------

Axes formatoptions

<i>xlim</i>	Set the x-axis limits
<i>ylim</i>	Set the y-axis limits
<i>axiscolor</i>	Color the x- and y-axes
<i>grid</i>	Display the grid
<i>tight</i>	Automatically adjust the plots.
<i>transpose</i>	Switch x- and y-axes

Axis tick formatoptions

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

Color coding formatoptions

<i>color</i>	Set the color coding
--------------	----------------------

Masking formatoptions

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

Label formatoptions

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

Post processing formatoptions

<code>post</code>	Apply your own postprocessing script
<code>post_timing</code>	Determine when to run the <code>post</code> formatoption

Miscellaneous formatoptions

<code>legend</code>	Draw a legend
<code>legendlabels</code>	Set the labels of the arrays in the legend
<code>sym_lims</code>	Make x- and y-axis symmetric
<code>ticksize</code>	Change the ticksize of the ticklabels
<code>tickweight</code>	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.cmmaps' key of the `psypplot.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 `psypplot.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 `psypplot.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 `psypplot.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 appearance (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 `psypplot.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
<code>'b'</code>	blue
<code>'g'</code>	green
<code>'r'</code>	red
<code>'c'</code>	cyan
<code>'m'</code>	magenta
<code>'y'</code>	yellow
<code>'k'</code>	black
<code>'w'</code>	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**

<i>data_dependent</i>	<code>bool(x) -> bool</code>
<i>transpose</i>	transpose Formatoption instance in the plotter
<i>xticks</i>	xticks Formatoption instance in the plotter
<i>yticklabels</i>	yticklabels Formatoption instance in the plotter

Methods

<i>update_axis</i> (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 `reParams` '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**

<code>array</code>	The numpy array of the data
<code>plot</code>	plot Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>yticks</code>	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**

<code>array</code>	The numpy array of the data
<code>plot</code>	plot Formatoption instance in the plotter
<code>sym_lims</code>	sym_lims Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>xticks</code>	xticks Formatoption instance in the plotter
<code>ylim</code>	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

<code>data_dependent</code>	<code>bool(x) -> bool</code>
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>yticks</code>	yticks Formatoption instance in the plotter

Methods

<code>update_axis(value)</code>

- *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 `reParams` '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

<code>array</code>	The numpy array of the data
<code>plot</code>	plot Formatoption instance in the plotter
<code>transpose</code>	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

<code>array</code>	The numpy array of the data
<code>plot</code>	plot Formatoption instance in the plotter
<code>sym_lims</code>	sym_lims Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>xlim</code>	xlim Formatoption instance in the plotter
<code>yticks</code>	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**

<code>children</code>	list() -> new empty list
<code>group</code>	str(object='') -> str
<code>name</code>	str(object='') -> str
<code>yticklabels</code>	yticklabels Formatoption instance in the plotter

Methods

<code>update(value)</code>	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**

<code>axis</code>	
<code>axisname</code>	<code>str(object='') -> str</code>
<code>name</code>	<code>str(object='') -> str</code>

- *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

<i>axis</i>	
<i>children</i>	list() -> new empty list
<i>data</i>	The data that is plotted
<i>name</i>	str(object='') -> str
<i>plot</i>	plot Formatoption instance in the plotter
<i>transpose</i>	transpose Formatoption instance in the plotter
<i>yticks</i>	yticks Formatoption instance in the plotter

Methods

<i>initialize_plot</i> (*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

<code>data</code>	The data that is plotted
<code>plot</code>	plot Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>yticks</code>	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

<i>labelsize</i>	Set the size of both, x- and y-label
<i>labelweight</i>	Set the font size of both, x- and y-label
<i>xlabel</i>	Set the x-axis label
<i>ylabel</i>	Set the y-axis label

Miscellaneous formatoptions

<i>ticksize</i>	Change the ticksize of the ticklabels
<i>tickweight</i>	Change the fontweight of the ticks

Axes formatoptions

<i>transpose</i>	Switch x- and y-axes
------------------	----------------------

Axis tick formatoptions

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

Post processing formatoptions

<i>post</i>	Apply your own postprocessing script
<i>post_timing</i>	Determine when to run the <code>post</code> 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

<i>children</i>	list() -> new empty list
<i>enhanced_attrs</i>	The enhanced attributes of the array
<i>name</i>	str(object='') -> str
<i>transpose</i>	transpose Formatoption instance in the plotter
<i>ylabel</i>	ylabel Formatoption instance in the plotter

Methods

<i>initialize_plot</i> (value)	Method that is called when the plot is made the first time
<i>update</i> (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

<i>array</i>	The numpy array of the data
<i>axisname</i>	str(object='') -> str
<i>children</i>	list() -> new empty list
<i>connections</i>	list() -> new empty list
<i>dependencies</i>	list() -> new empty list
<i>name</i>	str(object='') -> str
<i>plot</i>	plot Formatoption instance in the plotter
<i>sym_lims</i>	sym_lims Formatoption instance in the plotter
<i>transpose</i>	transpose Formatoption instance in the plotter
<i>xticks</i>	xticks Formatoption instance in the plotter
<i>yylim</i>	yylim Formatoption instance in the plotter

Methods

<i>initialize_plot</i> (<i>value</i>)	Method that is called when the plot is made the first time
<i>set_limit</i> (*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

<code>array</code>	The numpy array of the data
<code>plot</code>	plot Formatoption instance in the plotter
<code>sym_lims</code>	sym_lims Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>xticks</code>	xticks Formatoption instance in the plotter
<code>ylim</code>	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

<code>children</code>	<code>list()</code> -> new empty list
<code>group</code>	<code>str(object='')</code> -> str
<code>name</code>	<code>str(object='')</code> -> str
<code>yticklabels</code>	yticklabels Formatoption instance in the plotter

Methods

<code>update(value)</code>	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

<i>axis</i>	The axis on the axes to modify the ticks of
<i>dependencies</i>	list() -> new empty list
<i>name</i>	str(object='') -> str
<i>transpose</i>	transpose Formatoption instance in the plotter
<i>yticks</i>	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

<i>axis</i>	
<i>axisname</i>	str(object='') -> str
<i>name</i>	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

<i>axis</i>	
<i>data</i>	The data that is plotted
<i>name</i>	<code>str(object='') -> str</code>
<i>plot</i>	plot Formatoption instance in the plotter
<i>transpose</i>	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

<code>data</code>	The data that is plotted
<code>plot</code>	plot Formatoption instance in the plotter
<code>transpose</code>	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

<code>children</code>	<code>list()</code> -> new empty list
<code>enhanced_attrs</code>	The enhanced attributes of the array
<code>name</code>	<code>str(object='')</code> -> str
<code>transpose</code>	transpose <code>Formatoption</code> instance in the plotter

Methods

<code>initialize_plot(value)</code>	Method that is called when the plot is made the first time
<code>update(value)</code>	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

<code>array</code>	The numpy array of the data
<code>axisname</code>	<code>str(object='') -> str</code>
<code>children</code>	<code>list()</code> -> new empty list
<code>connections</code>	<code>list()</code> -> new empty list
<code>dependencies</code>	<code>list()</code> -> new empty list
<code>name</code>	<code>str(object='') -> str</code>
<code>plot</code>	plot Formatoption instance in the plotter
<code>sym_lims</code>	sym_lims Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>xlim</code>	xlim Formatoption instance in the plotter
<code>yticks</code>	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

<code>array</code>	The numpy array of the data
<code>plot</code>	plot Formatoption instance in the plotter
<code>sym_lims</code>	sym_lims Formatoption instance in the plotter
<code>transpose</code>	transpose Formatoption instance in the plotter
<code>xlim</code>	xlim Formatoption instance in the plotter
<code>yticks</code>	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

<code>BoundsValidator(*args, **kwargs)</code>	For parameter description see <code>matplotlib.rcsetup.ValidateInStrings</code> .
<code>DictValValidator(key, valid, validators, default)</code>	A validation class for formatoptions that expect dictionaries as values

Continued on next page

Table 335 – continued from previous page

<code>LineWidthValidator(key, valid[, ignorecase])</code>	valid is a list of legal strings
<code>TicksValidator(key, valid[, ignorecase])</code>	valid is a list of legal strings
<code>ValidateList([dtype, length, listtype])</code>	Validate a list of the specified <i>dtype</i>

Functions

<code>get_versions([requirements])</code>	
<code>patch_prior_1_0(plotter_d, versions)</code>	Patch psy_simple plotters for versions smaller than 1.0
<code>try_and_error(*funcs)</code>	Apply multiple validation functions
<code>validate_alpha(val)</code>	Validate an alpha value between 0 and 1
<code>validate_axiscolor(value)</code>	Validate a dictionary containing axiscolor definitions
<code>validate_cbarpos(value)</code>	Validate a colorbar position
<code>validate_cmap(val)</code>	Validate a colormap
<code>validate_cmmaps(cmmaps)</code>	Validate a dictionary of color lists
<code>validate_dataarray(val)</code>	
<code>validate_err_calc(val)</code>	Validation function for the
<code>validate_float(s)</code>	convert <i>s</i> to float or raise
<code>validate_fontweight(value)</code>	
<code>validate_iter(value)</code>	Validate that the given value is an iterable
<code>validate_legend(value)</code>	
<code>validate_limits(value)</code>	
<code>validate_lineplot(value)</code>	Validate the value for the LinePlotter.plot formatoption
<code>validate_marker(val)</code>	Does not really make a validation because markers can be quite of
<code>validate_none(b)</code>	Validate that None is given
<code>validate_str(s)</code>	Validate a string
<code>validate_sym_lims(val)</code>	
<code>validate_text(value)</code>	Validate a text formatoption
<code>validate_ticklabels(value)</code>	

Data

<code>patches</code>	patches to apply when loading a project
<code>rcParams</code>	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

<code>instance_check(val)</code>

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

<i>None</i>	data type (e.g. <code>float</code>) 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

Return type

Validate an alpha value between 0 and 1

Validate a dictionary containing axiscolor definitions

Returns

Raises `ValueError`

Validate a colorbar position

Returns list of strings with possible colorbar positions

Raises `ValueError`

Validate a colormap

Returns

Raises ValueError

Validate a dictionary of color lists

Raises `ValueError` – If one of the values in *cmaps* is not a color list

For all items (listname, list) in *cmaps*, the reversed list is automatically inserted with the `listname + '_r'` key.

Validation function for the `psy_simple.plotter.FldmeanPlotter.err_calc` formatoption

convert *s* to float or raise

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/>

p

- `psy_simple`, [30](#)
- `psy_simple.base`, [30](#)
- `psy_simple.colors`, [58](#)
- `psy_simple.plotters`, [62](#)
- `psy_simple.plugin`, [446](#)
- `psy_simple.version`, [450](#)

A

- `add2format_coord()` (`psy_simple.plotters.Plot2D` method), 285
- `add2format_coord()` (`psy_simple.plotters.VectorPlot` method), 390
- `allowed_dims` (`psy_simple.plotters.DensityPlotter` attribute), 186
- `allowed_dims` (`psy_simple.plotters.FldmeanPlotter` attribute), 220
- `allowed_dims` (`psy_simple.plotters.Simple2DBase` attribute), 294
- `allowed_dims` (`psy_simple.plotters.SimplePlotterBase` attribute), 328
- `allowed_vars` (`psy_simple.plotters.DensityPlotter` attribute), 186
- `allowed_vars` (`psy_simple.plotters.LinePlotter` attribute), 258
- `allowed_vars` (`psy_simple.plotters.SimplePlotterBase` attribute), 328
- `alpha` (`psy_simple.plotters.BarPlot` attribute), 73
- `alpha` (`psy_simple.plotters.BarPlotter` attribute), 76
- `AlternativeXCoord` (class in `psy_simple.plotters`), 64
- `AlternativeXCoordPost` (class in `psy_simple.plotters`), 67
- `array` (`psy_simple.plotters.BarXlim` attribute), 98
- `array` (`psy_simple.plotters.BarYlim` attribute), 101
- `array` (`psy_simple.plotters.DataGrid` attribute), 180
- `array` (`psy_simple.plotters.DataTicksCalculator` attribute), 183
- `array` (`psy_simple.plotters.Hist2DXRange` attribute), 241
- `array` (`psy_simple.plotters.Hist2DYRange` attribute), 243
- `array` (`psy_simple.plotters.Plot2D` attribute), 285
- `array` (`psy_simple.plotters.SimplePlot2D` attribute), 325
- `array` (`psy_simple.plotters.VectorBounds` attribute), 383
- `array` (`psy_simple.plotters.VectorCTicks` attribute), 385
- `array` (`psy_simple.plotters.ViolinXlim` attribute), 413
- `array` (`psy_simple.plotters.ViolinXTicks` attribute), 413
- `array` (`psy_simple.plotters.ViolinYlim` attribute), 416
- `array` (`psy_simple.plotters.ViolinYTicks` attribute), 415
- `array` (`psy_simple.plotters.Xlim` attribute), 435
- `array` (`psy_simple.plotters.Xlim2D` attribute), 436
- `array` (`psy_simple.plotters.Ylim` attribute), 444
- `array` (`psy_simple.plotters.Ylim2D` attribute), 445
- `ArrowSize` (class in `psy_simple.plotters`), 68
- `arrowsize` (`psy_simple.plotters.BaseVectorPlotter` attribute), 111
- `arrowsize` (`psy_simple.plotters.CombinedBase` attribute), 136
- `arrowsize` (`psy_simple.plotters.CombinedSimplePlotter` attribute), 175
- `arrowsize` (`psy_simple.plotters.CombinedVectorPlot` attribute), 177
- `arrowsize` (`psy_simple.plotters.SimpleVectorPlot` attribute), 345
- `arrowsize` (`psy_simple.plotters.SimpleVectorPlotter` attribute), 368
- `arrowsize` (`psy_simple.plotters.VectorPlot` attribute), 390
- `ArrowStyle` (class in `psy_simple.plotters`), 69
- `arrowstyle` (`psy_simple.plotters.BaseVectorPlotter` attribute), 111
- `arrowstyle` (`psy_simple.plotters.CombinedBase` attribute), 136
- `arrowstyle` (`psy_simple.plotters.CombinedSimplePlotter` attribute), 175
- `arrowstyle` (`psy_simple.plotters.CombinedVectorPlot` attribute), 178
- `arrowstyle` (`psy_simple.plotters.SimpleVectorPlot` attribute), 345
- `arrowstyle` (`psy_simple.plotters.SimpleVectorPlotter` attribute), 368
- `arrowstyle` (`psy_simple.plotters.VectorPlot` attribute), 390
- `axis` (`psy_simple.plotters.CbarOptions` attribute), 132
- `axis` (`psy_simple.plotters.TickLabelsBase` attribute), 373
- `axis` (`psy_simple.plotters.TicksBase` attribute), 378
- `axis` (`psy_simple.plotters.XTickLabels` attribute), 418
- `axis` (`psy_simple.plotters.XTickProps` attribute), 419
- `axis` (`psy_simple.plotters.XTicks` attribute), 421
- `axis` (`psy_simple.plotters.YTickLabels` attribute), 438
- `axis` (`psy_simple.plotters.YTickProps` attribute), 439
- `axis` (`psy_simple.plotters.YTicks` attribute), 440

axis_locations (psy_simple.plotters.CbarOptions attribute), 132
 axis_locations (psy_simple.plotters.CLabel attribute), 122
 AxisColor (class in psy_simple.plotters), 71
 axiscolor (psy_simple.plotters.BarPlotter attribute), 87
 axiscolor (psy_simple.plotters.CombinedSimplePlotter attribute), 166
 axiscolor (psy_simple.plotters.DensityPlotter attribute), 202
 axiscolor (psy_simple.plotters.FldmeanPlotter attribute), 230
 axiscolor (psy_simple.plotters.LinePlotter attribute), 269
 axiscolor (psy_simple.plotters.Simple2DPlotter attribute), 319
 axiscolor (psy_simple.plotters.SimplePlotterBase attribute), 328
 axiscolor (psy_simple.plotters.SimpleVectorPlotter attribute), 364
 axiscolor (psy_simple.plotters.ViolinPlotter attribute), 406
 axisname (psy_simple.plotters.CbarOptions attribute), 133
 axisname (psy_simple.plotters.TickPropsBase attribute), 374
 axisname (psy_simple.plotters.Xlim attribute), 435
 axisname (psy_simple.plotters.XTickProps attribute), 420
 axisname (psy_simple.plotters.Ylim attribute), 444
 axisname (psy_simple.plotters.YTickProps attribute), 439

B

BarAlpha (class in psy_simple.plotters), 72
 BarPlot (class in psy_simple.plotters), 73
 barplot (psyplot.project.plot attribute), 10
 BarPlotter (class in psy_simple.plotters), 74
 BarWidths (class in psy_simple.plotters), 94
 BarXlabel (class in psy_simple.plotters), 96
 BarXlim (class in psy_simple.plotters), 97
 BarXTickLabels (class in psy_simple.plotters), 95
 BarYlabel (class in psy_simple.plotters), 99
 BarYlim (class in psy_simple.plotters), 100
 BarYTickLabels (class in psy_simple.plotters), 98
 Base2D (class in psy_simple.plotters), 101
 BasePlotter (class in psy_simple.base), 31
 BaseVectorPlotter (class in psy_simple.plotters), 109
 bins (psy_simple.plotters.DensityPlotter attribute), 186
 bins (psy_simple.plotters.PointDensity attribute), 287
 Bounds (class in psy_simple.plotters), 119
 bounds (psy_simple.plotters.Base2D attribute), 103
 bounds (psy_simple.plotters.BaseVectorPlotter attribute), 111
 bounds (psy_simple.plotters.Cbar attribute), 131
 bounds (psy_simple.plotters.CMap attribute), 123

bounds (psy_simple.plotters.CombinedBase attribute), 142
 bounds (psy_simple.plotters.CombinedSimplePlotter attribute), 150
 bounds (psy_simple.plotters.CombinedVectorPlot attribute), 178
 bounds (psy_simple.plotters.CTicks attribute), 129
 bounds (psy_simple.plotters.DensityPlotter attribute), 190
 bounds (psy_simple.plotters.Plot2D attribute), 285
 bounds (psy_simple.plotters.ScalarCombinedBase attribute), 288
 bounds (psy_simple.plotters.Simple2DBase attribute), 294
 bounds (psy_simple.plotters.Simple2DPlotter attribute), 307
 bounds (psy_simple.plotters.SimplePlot2D attribute), 325
 bounds (psy_simple.plotters.SimpleVectorPlot attribute), 345
 bounds (psy_simple.plotters.SimpleVectorPlotter attribute), 352
 bounds (psy_simple.plotters.VectorCbar attribute), 386
 bounds (psy_simple.plotters.VectorColor attribute), 388
 bounds (psy_simple.plotters.VectorCTicks attribute), 385
 bounds (psy_simple.plotters.VectorPlot attribute), 390
 BoundsValidator (class in psy_simple.plugin), 447

C

Cbar (class in psy_simple.plotters), 129
 cbar (psy_simple.plotters.Base2D attribute), 104
 cbar (psy_simple.plotters.BaseVectorPlotter attribute), 112
 cbar (psy_simple.plotters.Bounds attribute), 120
 cbar (psy_simple.plotters.CbarOptions attribute), 133
 cbar (psy_simple.plotters.CbarSpacing attribute), 134
 cbar (psy_simple.plotters.CLabel attribute), 122
 cbar (psy_simple.plotters.CMap attribute), 123
 cbar (psy_simple.plotters.CombinedBase attribute), 143
 cbar (psy_simple.plotters.CombinedSimplePlotter attribute), 151
 cbar (psy_simple.plotters.ContourLevels attribute), 179
 cbar (psy_simple.plotters.CTickLabels attribute), 124
 cbar (psy_simple.plotters.CTickProps attribute), 125
 cbar (psy_simple.plotters.CTicks attribute), 129
 cbar (psy_simple.plotters.CTickSize attribute), 126
 cbar (psy_simple.plotters.CTickWeight attribute), 128
 cbar (psy_simple.plotters.DensityPlotter attribute), 191
 cbar (psy_simple.plotters.ScalarCombinedBase attribute), 289
 cbar (psy_simple.plotters.Simple2DBase attribute), 295
 cbar (psy_simple.plotters.Simple2DPlotter attribute), 308
 cbar (psy_simple.plotters.SimpleVectorPlotter attribute), 353
 cbar (psy_simple.plotters.VCLabel attribute), 382

- cbar (psy_simple.plotters.VectorBounds attribute), 384
- cbar (psy_simple.plotters.VectorCTicks attribute), 385
- CbarOptions (class in psy_simple.plotters), 132
- CbarSpacing (class in psy_simple.plotters), 133
- cbarspacing (psy_simple.plotters.Base2D attribute), 104
- cbarspacing (psy_simple.plotters.BaseVectorPlotter attribute), 115
- cbarspacing (psy_simple.plotters.Cbar attribute), 131
- cbarspacing (psy_simple.plotters.CombinedSimplePlotter attribute), 151
- cbarspacing (psy_simple.plotters.DensityPlotter attribute), 191
- cbarspacing (psy_simple.plotters.Simple2DBase attribute), 296
- cbarspacing (psy_simple.plotters.Simple2DPlotter attribute), 309
- cbarspacing (psy_simple.plotters.SimpleVectorPlotter attribute), 353
- cbarspacing (psy_simple.plotters.VectorCbar attribute), 387
- cbounds (psy_simple.plotters.ContourLevels attribute), 179
- check_data() (psy_simple.plotters.BaseVectorPlotter class method), 112
- check_data() (psy_simple.plotters.CombinedBase class method), 136
- check_data() (psy_simple.plotters.Simple2DBase class method), 294
- check_data() (psy_simple.plotters.SimplePlotterBase class method), 329
- children (psy_simple.plotters.BarPlot attribute), 74
- children (psy_simple.plotters.CbarOptions attribute), 133
- children (psy_simple.plotters.CLabel attribute), 122
- children (psy_simple.plotters.CTickProps attribute), 125
- children (psy_simple.plotters.DataGrid attribute), 180
- children (psy_simple.plotters.ErrorCalculator attribute), 215
- children (psy_simple.plotters.ErrorPlot attribute), 216
- children (psy_simple.plotters.LabelOptions attribute), 246
- children (psy_simple.plotters.LabelProps attribute), 247
- children (psy_simple.plotters.LimitBase attribute), 253
- children (psy_simple.plotters.LinePlot attribute), 255
- children (psy_simple.plotters.Plot2D attribute), 285
- children (psy_simple.plotters.VectorPlot attribute), 390
- children (psy_simple.plotters.ViolinPlot attribute), 392
- children (psy_simple.plotters.Xlabel attribute), 433
- children (psy_simple.plotters.Xlim attribute), 435
- children (psy_simple.plotters.XRotation attribute), 417
- children (psy_simple.plotters.XTicks attribute), 421
- children (psy_simple.plotters.Ylabel attribute), 443
- children (psy_simple.plotters.Ylim attribute), 444
- children (psy_simple.plotters.YRotation attribute), 437
- CLabel (class in psy_simple.plotters), 120
- clabel (psy_simple.plotters.Base2D attribute), 104
- clabel (psy_simple.plotters.BaseVectorPlotter attribute), 116
- clabel (psy_simple.plotters.CombinedSimplePlotter attribute), 157
- clabel (psy_simple.plotters.DensityPlotter attribute), 194
- clabel (psy_simple.plotters.Simple2DBase attribute), 297
- clabel (psy_simple.plotters.Simple2DPlotter attribute), 311
- clabel (psy_simple.plotters.SimpleVectorPlotter attribute), 356
- clabelprops (psy_simple.plotters.Base2D attribute), 105
- clabelprops (psy_simple.plotters.BaseVectorPlotter attribute), 116
- clabelprops (psy_simple.plotters.CombinedSimplePlotter attribute), 158
- clabelprops (psy_simple.plotters.DensityPlotter attribute), 195
- clabelprops (psy_simple.plotters.Simple2DBase attribute), 298
- clabelprops (psy_simple.plotters.Simple2DPlotter attribute), 312
- clabelprops (psy_simple.plotters.SimpleVectorPlotter attribute), 356
- clabelsize (psy_simple.plotters.Base2D attribute), 105
- clabelsize (psy_simple.plotters.BaseVectorPlotter attribute), 117
- clabelsize (psy_simple.plotters.CombinedSimplePlotter attribute), 158
- clabelsize (psy_simple.plotters.DensityPlotter attribute), 195
- clabelsize (psy_simple.plotters.Simple2DBase attribute), 298
- clabelsize (psy_simple.plotters.Simple2DPlotter attribute), 312
- clabelsize (psy_simple.plotters.SimpleVectorPlotter attribute), 357
- clabelweight (psy_simple.plotters.Base2D attribute), 105
- clabelweight (psy_simple.plotters.BaseVectorPlotter attribute), 117
- clabelweight (psy_simple.plotters.CombinedSimplePlotter attribute), 158
- clabelweight (psy_simple.plotters.DensityPlotter attribute), 195
- clabelweight (psy_simple.plotters.Simple2DBase attribute), 298
- clabelweight (psy_simple.plotters.Simple2DPlotter attribute), 312
- clabelweight (psy_simple.plotters.SimpleVectorPlotter attribute), 357
- clear_other_texts() (psy_simple.base.Figtitle method), 39
- CMap (class in psy_simple.plotters), 122
- cmap (psy_simple.plotters.Base2D attribute), 106
- cmap (psy_simple.plotters.BaseVectorPlotter attribute),

- 115
- cmap (psy_simple.plotters.Bounds attribute), 120
- cmap (psy_simple.plotters.Cbar attribute), 131
- cmap (psy_simple.plotters.CombinedSimplePlotter attribute), 151
- cmap (psy_simple.plotters.CombinedVectorPlot attribute), 178
- cmap (psy_simple.plotters.ContourLevels attribute), 179
- cmap (psy_simple.plotters.DensityPlotter attribute), 191
- cmap (psy_simple.plotters.Plot2D attribute), 285
- cmap (psy_simple.plotters.Simple2DBase attribute), 296
- cmap (psy_simple.plotters.Simple2DPlotter attribute), 309
- cmap (psy_simple.plotters.SimplePlot2D attribute), 325
- cmap (psy_simple.plotters.SimpleVectorPlot attribute), 346
- cmap (psy_simple.plotters.SimpleVectorPlotter attribute), 353
- cmap (psy_simple.plotters.VectorBounds attribute), 384
- cmap (psy_simple.plotters.VectorCbar attribute), 387
- cmap (psy_simple.plotters.VectorColor attribute), 388
- cmap (psy_simple.plotters.VectorPlot attribute), 390
- color (psy_simple.plotters.BarPlot attribute), 74
- color (psy_simple.plotters.BarPlotter attribute), 80
- color (psy_simple.plotters.BaseVectorPlotter attribute), 113
- color (psy_simple.plotters.CombinedBase attribute), 136
- color (psy_simple.plotters.CombinedSimplePlotter attribute), 151
- color (psy_simple.plotters.CombinedVectorPlot attribute), 178
- color (psy_simple.plotters.ErrorPlot attribute), 216
- color (psy_simple.plotters.FldmeanPlotter attribute), 222
- color (psy_simple.plotters.Legend attribute), 251
- color (psy_simple.plotters.LinePlot attribute), 255
- color (psy_simple.plotters.LinePlotter attribute), 261
- color (psy_simple.plotters.Simple2DPlotter attribute), 303
- color (psy_simple.plotters.SimplePlotterBase attribute), 329
- color (psy_simple.plotters.SimpleVectorPlot attribute), 346
- color (psy_simple.plotters.SimpleVectorPlotter attribute), 354
- color (psy_simple.plotters.VectorBounds attribute), 384
- color (psy_simple.plotters.VectorCbar attribute), 387
- color (psy_simple.plotters.VectorCTicks attribute), 385
- color (psy_simple.plotters.VectorPlot attribute), 390
- color (psy_simple.plotters.ViolinPlot attribute), 392
- color (psy_simple.plotters.ViolinPlotter attribute), 398
- colorbar (psy_simple.plotters.CbarOptions attribute), 133
- combined (psyplot.project.plot attribute), 8
- CombinedBase (class in psy_simple.plotters), 134
- CombinedSimplePlotter (class in psy_simple.plotters), 146
- CombinedVectorPlot (class in psy_simple.plotters), 177
- connections (psy_simple.plotters.Bounds attribute), 120
- connections (psy_simple.plotters.CbarSpacing attribute), 134
- connections (psy_simple.plotters.CMap attribute), 123
- connections (psy_simple.plotters.DataPrecision attribute), 182
- connections (psy_simple.plotters.ErrorAlpha attribute), 213
- connections (psy_simple.plotters.Extend attribute), 217
- connections (psy_simple.plotters.LimitBase attribute), 253
- connections (psy_simple.plotters.LineWidth attribute), 278
- connections (psy_simple.plotters.MarkerSize attribute), 279
- connections (psy_simple.plotters.MissColor attribute), 282
- connections (psy_simple.plotters.VectorPlot attribute), 390
- connections (psy_simple.plotters.Xlim attribute), 435
- connections (psy_simple.plotters.Ylim attribute), 444
- ContourLevels (class in psy_simple.plotters), 178
- convert_radian (psy_simple.plotters.Base2D attribute), 106
- coord (psy_simple.plotters.BarPlotter attribute), 76
- coord (psy_simple.plotters.DensityPlotter attribute), 187
- coord (psy_simple.plotters.FldmeanPlotter attribute), 220
- coord (psy_simple.plotters.Hist2DXRange attribute), 241
- coord (psy_simple.plotters.Hist2DXYRange attribute), 243
- coord (psy_simple.plotters.LinePlotter attribute), 258
- coord (psy_simple.plotters.PointDensity attribute), 287
- CTickLabels (class in psy_simple.plotters), 124
- cticklabels (psy_simple.plotters.Base2D attribute), 106
- cticklabels (psy_simple.plotters.BaseVectorPlotter attribute), 118
- cticklabels (psy_simple.plotters.CombinedSimplePlotter attribute), 169
- cticklabels (psy_simple.plotters.DensityPlotter attribute), 205
- cticklabels (psy_simple.plotters.Simple2DBase attribute), 298
- cticklabels (psy_simple.plotters.Simple2DPlotter attribute), 321
- cticklabels (psy_simple.plotters.SimpleVectorPlotter attribute), 366
- CTickProps (class in psy_simple.plotters), 125
- ctickprops (psy_simple.plotters.Base2D attribute), 106
- ctickprops (psy_simple.plotters.BaseVectorPlotter attribute), 115
- ctickprops (psy_simple.plotters.CombinedSimplePlotter attribute), 152

ctickprops (psy_simple.plotters.CTickSize attribute), 126
 ctickprops (psy_simple.plotters.CTickWeight attribute), 128
 ctickprops (psy_simple.plotters.DensityPlotter attribute), 191
 ctickprops (psy_simple.plotters.Simple2DBase attribute), 296
 ctickprops (psy_simple.plotters.Simple2DPlotter attribute), 309
 ctickprops (psy_simple.plotters.SimpleVectorPlotter attribute), 354
 CTicks (class in psy_simple.plotters), 128
 cticks (psy_simple.plotters.Base2D attribute), 106
 cticks (psy_simple.plotters.BaseVectorPlotter attribute), 113
 cticks (psy_simple.plotters.CombinedBase attribute), 145
 cticks (psy_simple.plotters.CombinedSimplePlotter attribute), 169
 cticks (psy_simple.plotters.DensityPlotter attribute), 205
 cticks (psy_simple.plotters.ScalarCombinedBase attribute), 290
 cticks (psy_simple.plotters.Simple2DBase attribute), 299
 cticks (psy_simple.plotters.Simple2DPlotter attribute), 321
 cticks (psy_simple.plotters.SimpleVectorPlotter attribute), 366
 CTickSize (class in psy_simple.plotters), 126
 cticksize (psy_simple.plotters.Base2D attribute), 107
 cticksize (psy_simple.plotters.BaseVectorPlotter attribute), 115
 cticksize (psy_simple.plotters.CombinedSimplePlotter attribute), 152
 cticksize (psy_simple.plotters.DensityPlotter attribute), 192
 cticksize (psy_simple.plotters.Simple2DBase attribute), 296
 cticksize (psy_simple.plotters.Simple2DPlotter attribute), 309
 cticksize (psy_simple.plotters.SimpleVectorPlotter attribute), 354
 CTickWeight (class in psy_simple.plotters), 127
 ctickweight (psy_simple.plotters.Base2D attribute), 107
 ctickweight (psy_simple.plotters.BaseVectorPlotter attribute), 115
 ctickweight (psy_simple.plotters.CombinedSimplePlotter attribute), 152
 ctickweight (psy_simple.plotters.DensityPlotter attribute), 192
 ctickweight (psy_simple.plotters.Simple2DBase attribute), 297
 ctickweight (psy_simple.plotters.Simple2DPlotter attribute), 310
 ctickweight (psy_simple.plotters.SimpleVectorPlotter attribute), 354

D

data (psy_simple.plotters.CbarOptions attribute), 133
 data (psy_simple.plotters.XTicks attribute), 421
 data (psy_simple.plotters.XTicks2D attribute), 423
 data (psy_simple.plotters.YTicks attribute), 440
 data (psy_simple.plotters.YTicks2D attribute), 441
 data_dependent (psy_simple.base.TextBase attribute), 46
 data_dependent (psy_simple.base.ValueMaskBase attribute), 56
 data_dependent (psy_simple.plotters.AlternativeXCoord attribute), 66
 data_dependent (psy_simple.plotters.BarPlot attribute), 74
 data_dependent (psy_simple.plotters.CLabel attribute), 122
 data_dependent (psy_simple.plotters.CombinedVectorPlot attribute), 178
 data_dependent (psy_simple.plotters.DataPrecision attribute), 182
 data_dependent (psy_simple.plotters.DataTicksCalculator attribute), 183
 data_dependent (psy_simple.plotters.Density attribute), 183
 data_dependent (psy_simple.plotters.ErrorCalculator attribute), 215
 data_dependent (psy_simple.plotters.ErrorPlot attribute), 216
 data_dependent (psy_simple.plotters.Hist2DXRange attribute), 241
 data_dependent (psy_simple.plotters.Hist2DYRange attribute), 243
 data_dependent (psy_simple.plotters.HistBins attribute), 244
 data_dependent (psy_simple.plotters.LegendLabels attribute), 252
 data_dependent (psy_simple.plotters.LinePlot attribute), 255
 data_dependent (psy_simple.plotters.MeanCalculator attribute), 281
 data_dependent (psy_simple.plotters.NormedHist2D attribute), 283
 data_dependent (psy_simple.plotters.Plot2D attribute), 285
 data_dependent (psy_simple.plotters.PointDensity attribute), 287
 data_dependent (psy_simple.plotters.SimplePlot2D attribute), 325
 data_dependent (psy_simple.plotters.SimpleVectorPlot attribute), 346
 data_dependent (psy_simple.plotters.VectorCalculator attribute), 385
 data_dependent (psy_simple.plotters.VectorPlot attribute), 390

- data_dependent (psy_simple.plotters.ViolinPlot attribute), 392
- data_dependent (psy_simple.plotters.ViolinXTickLabels attribute), 411
- data_dependent (psy_simple.plotters.ViolinYTickLabels attribute), 414
- data_iterator (psy_simple.plotters.AlternativeXCoord attribute), 66
- DataGrid (class in psy_simple.plotters), 179
- datagrid (psy_simple.plotters.Base2D attribute), 108
- datagrid (psy_simple.plotters.BaseVectorPlotter attribute), 119
- datagrid (psy_simple.plotters.CombinedSimplePlotter attribute), 175
- datagrid (psy_simple.plotters.DensityPlotter attribute), 210
- datagrid (psy_simple.plotters.Simple2DBase attribute), 300
- datagrid (psy_simple.plotters.Simple2DPlotter attribute), 324
- datagrid (psy_simple.plotters.SimpleVectorPlotter attribute), 369
- DataPrecision (class in psy_simple.plotters), 181
- DataTicksCalculator (class in psy_simple.plotters), 182
- default_formatters (psy_simple.plotters.CTickLabels attribute), 124
- default_locator (psy_simple.plotters.CTicks attribute), 129
- delimiter (psy_simple.base.TextBase attribute), 46
- Density (class in psy_simple.plotters), 183
- density (psy_simple.plotters.BaseVectorPlotter attribute), 114
- density (psy_simple.plotters.CombinedSimplePlotter attribute), 149
- density (psy_simple.plotters.CombinedVectorPlot attribute), 178
- density (psy_simple.plotters.DataPrecision attribute), 182
- density (psy_simple.plotters.DensityPlotter attribute), 188
- density (psy_simple.plotters.SimpleVectorPlot attribute), 346
- density (psy_simple.plotters.SimpleVectorPlotter attribute), 369
- density (psy_simple.plotters.VectorPlot attribute), 390
- density (psyplot.project.plot attribute), 9
- DensityPlotter (class in psy_simple.plotters), 184
- dependencies (psy_simple.plotters.ArrowSize attribute), 69
- dependencies (psy_simple.plotters.ArrowStyle attribute), 70
- dependencies (psy_simple.plotters.BarPlot attribute), 74
- dependencies (psy_simple.plotters.BarXlim attribute), 98
- dependencies (psy_simple.plotters.BarXTickLabels attribute), 96
- dependencies (psy_simple.plotters.BarYlim attribute), 101
- dependencies (psy_simple.plotters.BarYTickLabels attribute), 99
- dependencies (psy_simple.plotters.Cbar attribute), 131
- dependencies (psy_simple.plotters.CbarOptions attribute), 133
- dependencies (psy_simple.plotters.CLabel attribute), 122
- dependencies (psy_simple.plotters.ContourLevels attribute), 179
- dependencies (psy_simple.plotters.CTicks attribute), 129
- dependencies (psy_simple.plotters.CTickSize attribute), 127
- dependencies (psy_simple.plotters.CTickWeight attribute), 128
- dependencies (psy_simple.plotters.DataPrecision attribute), 182
- dependencies (psy_simple.plotters.Density attribute), 183
- dependencies (psy_simple.plotters.Hist2DXRange attribute), 242
- dependencies (psy_simple.plotters.HistBins attribute), 244
- dependencies (psy_simple.plotters.Legend attribute), 251
- dependencies (psy_simple.plotters.MissColor attribute), 282
- dependencies (psy_simple.plotters.Plot2D attribute), 285
- dependencies (psy_simple.plotters.PointDensity attribute), 287
- dependencies (psy_simple.plotters.SimplePlot2D attribute), 326
- dependencies (psy_simple.plotters.SymmetricLimits attribute), 371
- dependencies (psy_simple.plotters.TickLabelsBase attribute), 373
- dependencies (psy_simple.plotters.TicksBase attribute), 378
- dependencies (psy_simple.plotters.TickSize attribute), 375
- dependencies (psy_simple.plotters.TickWeight attribute), 376
- dependencies (psy_simple.plotters.VectorCalculator attribute), 385
- dependencies (psy_simple.plotters.VectorCbar attribute), 387
- dependencies (psy_simple.plotters.VectorColor attribute), 388
- dependencies (psy_simple.plotters.VectorCTicks attribute), 385
- dependencies (psy_simple.plotters.Xlim attribute), 435
- dependencies (psy_simple.plotters.XTickLabels attribute), 418
- dependencies (psy_simple.plotters.Ylim attribute), 444
- dependencies (psy_simple.plotters.YTickLabels attribute), 438

DictValValidator (class in psy_simple.plugin), 447
 diff() (psy_simple.base.Text method), 45
 diff() (psy_simple.plotters.AlternativeXCoord method), 66
 draw_colorbar() (psy_simple.plotters.Cbar method), 131
 dtdata (psy_simple.plotters.DtTicksBase attribute), 212
 DtTicksBase (class in psy_simple.plotters), 211
 dtype (psy_simple.plugin.ValidateList attribute), 448

E

enhanced_attrs (psy_simple.base.Figtitle attribute), 40
 enhanced_attrs (psy_simple.base.TextBase attribute), 46
 enhanced_attrs (psy_simple.plotters.Xlabel attribute), 433
 enhanced_attrs (psy_simple.plotters.Ylabel attribute), 443
 err_calc (psy_simple.plotters.FldmeanPlotter attribute), 221
 error (psy_simple.plotters.ErrorAlpha attribute), 213
 error (psy_simple.plotters.FldmeanPlotter attribute), 238
 error (psy_simple.plotters.LinePlotter attribute), 259
 ErrorAlpha (class in psy_simple.plotters), 212
 erroralpha (psy_simple.plotters.FldmeanPlotter attribute), 223
 erroralpha (psy_simple.plotters.LinePlotter attribute), 260
 ErrorCalculator (class in psy_simple.plotters), 213
 ErrorPlot (class in psy_simple.plotters), 215
 Extend (class in psy_simple.plotters), 216
 extend (psy_simple.plotters.Base2D attribute), 108
 extend (psy_simple.plotters.BaseVectorPlotter attribute), 116
 extend (psy_simple.plotters.Cbar attribute), 131
 extend (psy_simple.plotters.CombinedSimplePlotter attribute), 153
 extend (psy_simple.plotters.DensityPlotter attribute), 192
 extend (psy_simple.plotters.Simple2DBase attribute), 297
 extend (psy_simple.plotters.Simple2DPlotter attribute), 310
 extend (psy_simple.plotters.SimpleVectorPlotter attribute), 355
 extend (psy_simple.plotters.VectorCbar attribute), 387

F

Figtitle (class in psy_simple.base), 38
 figtitle (psy_simple.base.BasePlotter attribute), 33
 figtitle (psy_simple.base.TitlesPlotter attribute), 51
 figtitle (psy_simple.plotters.BarPlotter attribute), 82
 figtitle (psy_simple.plotters.CombinedSimplePlotter attribute), 158
 figtitle (psy_simple.plotters.DensityPlotter attribute), 195
 figtitle (psy_simple.plotters.FldmeanPlotter attribute), 223
 figtitle (psy_simple.plotters.LinePlotter attribute), 262

figtitle (psy_simple.plotters.Simple2DPlotter attribute), 313
 figtitle (psy_simple.plotters.SimplePlotterBase attribute), 333
 figtitle (psy_simple.plotters.SimpleVectorPlotter attribute), 357
 figtitle (psy_simple.plotters.ViolinPlotter attribute), 399
 figtitleprops (psy_simple.base.BasePlotter attribute), 34
 figtitleprops (psy_simple.base.TitlesPlotter attribute), 52
 figtitleprops (psy_simple.plotters.BarPlotter attribute), 83
 figtitleprops (psy_simple.plotters.CombinedSimplePlotter attribute), 159
 figtitleprops (psy_simple.plotters.DensityPlotter attribute), 196
 figtitleprops (psy_simple.plotters.FldmeanPlotter attribute), 224
 figtitleprops (psy_simple.plotters.LinePlotter attribute), 263
 figtitleprops (psy_simple.plotters.Simple2DPlotter attribute), 313
 figtitleprops (psy_simple.plotters.SimplePlotterBase attribute), 334
 figtitleprops (psy_simple.plotters.SimpleVectorPlotter attribute), 358
 figtitleprops (psy_simple.plotters.ViolinPlotter attribute), 400
 figtitlesize (psy_simple.base.BasePlotter attribute), 34
 figtitlesize (psy_simple.base.TitlesPlotter attribute), 52
 figtitlesize (psy_simple.plotters.BarPlotter attribute), 83
 figtitlesize (psy_simple.plotters.CombinedSimplePlotter attribute), 159
 figtitlesize (psy_simple.plotters.DensityPlotter attribute), 196
 figtitlesize (psy_simple.plotters.FldmeanPlotter attribute), 224
 figtitlesize (psy_simple.plotters.LinePlotter attribute), 263
 figtitlesize (psy_simple.plotters.Simple2DPlotter attribute), 314
 figtitlesize (psy_simple.plotters.SimplePlotterBase attribute), 334
 figtitlesize (psy_simple.plotters.SimpleVectorPlotter attribute), 358
 figtitlesize (psy_simple.plotters.ViolinPlotter attribute), 400
 figtitleweight (psy_simple.base.BasePlotter attribute), 34
 figtitleweight (psy_simple.base.TitlesPlotter attribute), 52
 figtitleweight (psy_simple.plotters.BarPlotter attribute), 83
 figtitleweight (psy_simple.plotters.CombinedSimplePlotter attribute), 160
 figtitleweight (psy_simple.plotters.DensityPlotter attribute), 197
 figtitleweight (psy_simple.plotters.FldmeanPlotter

attribute), 224
 figtitleweight (psy_simple.plotters.LinePlotter attribute), 263
 figtitleweight (psy_simple.plotters.Simple2DPlotter attribute), 314
 figtitleweight (psy_simple.plotters.SimplePlotterBase attribute), 335
 figtitleweight (psy_simple.plotters.SimpleVectorPlotter attribute), 358
 figtitleweight (psy_simple.plotters.ViolinPlotter attribute), 401
 figure_positions (psy_simple.plotters.Cbar attribute), 131
 finish_update() (psy_simple.base.Text method), 45
 finish_update() (psy_simple.plotters.Cbar method), 131
 FixedBoundaryNorm (class in psy_simple.colors), 58
 FixedColorMap (class in psy_simple.colors), 59
 fldmean (psypplot.project.plot attribute), 11
 FldmeanPlotter (class in psy_simple.plotters), 217
 format_coord (psy_simple.plotters.Plot2D attribute), 285
 format_coord (psy_simple.plotters.VectorPlot attribute), 390
 format_coord_func() (in module psy_simple.plotters), 445
 from_list() (psy_simple.colors.FixedColorMap static method), 61
 full_array (psy_simple.plotters.DataTicksCalculator attribute), 183

G

get_alternative_coord() (psy_simple.plotters.AlternativeXCoord method), 66
 get_artists_and_labels() (psy_simple.plotters.Legend method), 251
 get_cmap() (in module psy_simple.colors), 61
 get_cmap() (psy_simple.plotters.CMap method), 123
 get_enhanced_attrs() (psy_simple.base.TextBase method), 46
 get_fig_data_attrs() (psy_simple.base.TextBase method), 46
 get_fmt_widget() (psy_simple.base.TextBase method), 47
 get_fmt_widget() (psy_simple.plotters.Bounds method), 120
 get_fmt_widget() (psy_simple.plotters.CMap method), 123
 get_locator() (psy_simple.plotters.TicksBase method), 378
 get_versions() (in module psy_simple.plugin), 448
 get_x() (psy_simple.plotters.Transpose method), 381
 get_xys() (psy_simple.plotters.BarPlot method), 74
 get_xyz_1d() (psy_simple.plotters.Plot2D method), 285
 get_xyz_1d() (psy_simple.plotters.VectorPlot method), 390
 get_xyz_2d() (psy_simple.plotters.Plot2D method), 285

get_xyz_2d() (psy_simple.plotters.VectorPlot method), 390
 get_xyz_tri() (psy_simple.plotters.Plot2D method), 285
 get_xyz_tri() (psy_simple.plotters.VectorPlot method), 390
 get_y() (psy_simple.plotters.Transpose method), 381
 Grid (class in psy_simple.plotters), 239
 grid (psy_simple.plotters.BarPlotter attribute), 88
 grid (psy_simple.plotters.CombinedSimplePlotter attribute), 167
 grid (psy_simple.plotters.DensityPlotter attribute), 203
 grid (psy_simple.plotters.FldmeanPlotter attribute), 230
 grid (psy_simple.plotters.LinePlotter attribute), 269
 grid (psy_simple.plotters.Simple2DPlotter attribute), 320
 grid (psy_simple.plotters.SimplePlotterBase attribute), 329
 grid (psy_simple.plotters.SimpleVectorPlotter attribute), 364
 grid (psy_simple.plotters.ViolinPlotter attribute), 407
 group (psy_simple.base.TextBase attribute), 47
 group (psy_simple.base.Tight attribute), 48
 group (psy_simple.base.ValueMaskBase attribute), 56
 group (psy_simple.plotters.AlternativeXCoord attribute), 66
 group (psy_simple.plotters.ArrowSize attribute), 69
 group (psy_simple.plotters.ArrowStyle attribute), 70
 group (psy_simple.plotters.AxisColor attribute), 72
 group (psy_simple.plotters.BarPlot attribute), 74
 group (psy_simple.plotters.Bounds attribute), 120
 group (psy_simple.plotters.Cbar attribute), 131
 group (psy_simple.plotters.CbarSpacing attribute), 134
 group (psy_simple.plotters.CLabel attribute), 122
 group (psy_simple.plotters.CMap attribute), 124
 group (psy_simple.plotters.CTickProps attribute), 125
 group (psy_simple.plotters.CTickSize attribute), 127
 group (psy_simple.plotters.CTickWeight attribute), 128
 group (psy_simple.plotters.DataPrecision attribute), 182
 group (psy_simple.plotters.Density attribute), 183
 group (psy_simple.plotters.ErrorAlpha attribute), 213
 group (psy_simple.plotters.ErrorCalculator attribute), 215
 group (psy_simple.plotters.ErrorPlot attribute), 216
 group (psy_simple.plotters.Extend attribute), 217
 group (psy_simple.plotters.Grid attribute), 240
 group (psy_simple.plotters.Hist2DXRange attribute), 242
 group (psy_simple.plotters.HistBins attribute), 244
 group (psy_simple.plotters.LabelProps attribute), 247
 group (psy_simple.plotters.LabelSize attribute), 248
 group (psy_simple.plotters.LabelWeight attribute), 249
 group (psy_simple.plotters.LimitBase attribute), 253
 group (psy_simple.plotters.LineColors attribute), 254
 group (psy_simple.plotters.LinePlot attribute), 255
 group (psy_simple.plotters.MeanCalculator attribute), 281
 group (psy_simple.plotters.MissColor attribute), 282

group (psy_simple.plotters.NormedHist2D attribute), 283
 group (psy_simple.plotters.Plot2D attribute), 285
 group (psy_simple.plotters.PointDensity attribute), 287
 group (psy_simple.plotters.TickLabelsBase attribute), 373
 group (psy_simple.plotters.TicksBase attribute), 378
 group (psy_simple.plotters.TicksManager attribute), 379
 group (psy_simple.plotters.Transpose attribute), 381
 group (psy_simple.plotters.VectorColor attribute), 388
 group (psy_simple.plotters.VectorPlot attribute), 390
 group (psy_simple.plotters.ViolinPlot attribute), 392
 group (psy_simple.plotters.XRotation attribute), 417
 group (psy_simple.plotters.YRotation attribute), 437

H

hist2d() (psy_simple.plotters.NormedHist2D method), 283
 Hist2DXRange (class in psy_simple.plotters), 240
 Hist2DYRange (class in psy_simple.plotters), 242
 HistBins (class in psy_simple.plotters), 243

I

init_kwargs (psy_simple.plotters.Cbar attribute), 131
 initialize_plot() (psy_simple.base.Figtitle method), 40
 initialize_plot() (psy_simple.base.Title method), 50
 initialize_plot() (psy_simple.plotters.AxisColor method), 72
 initialize_plot() (psy_simple.plotters.Cbar method), 131
 initialize_plot() (psy_simple.plotters.TickLabelsBase method), 373
 initialize_plot() (psy_simple.plotters.TicksBase method), 378
 initialize_plot() (psy_simple.plotters.Transpose method), 381
 initialize_plot() (psy_simple.plotters.Xlabel method), 433
 initialize_plot() (psy_simple.plotters.Xlim method), 435
 initialize_plot() (psy_simple.plotters.XTickLabels method), 418
 initialize_plot() (psy_simple.plotters.XTicks method), 421
 initialize_plot() (psy_simple.plotters.Ylabel method), 443
 instance_check() (psy_simple.plugin.BoundsValidator method), 447
 interp_bounds (psy_simple.plotters.CombinedSimplePlotter attribute), 175
 interp_bounds (psy_simple.plotters.DensityPlotter attribute), 210
 interp_bounds (psy_simple.plotters.Plot2D attribute), 285
 interp_bounds (psy_simple.plotters.Simple2DPlotter attribute), 303
 interp_bounds (psy_simple.plotters.SimplePlot2D attribute), 326
 InterpolateBounds (class in psy_simple.plotters), 244

L

label_props() (in module psy_simple.base), 56
 label_size() (in module psy_simple.base), 57
 label_weight() (in module psy_simple.base), 57
 LabelOptions (class in psy_simple.plotters), 245
 LabelProps (class in psy_simple.plotters), 246
 labelprops (psy_simple.plotters.BarPlotter attribute), 83
 labelprops (psy_simple.plotters.CombinedSimplePlotter attribute), 160
 labelprops (psy_simple.plotters.DensityPlotter attribute), 197
 labelprops (psy_simple.plotters.FldmeanPlotter attribute), 224
 labelprops (psy_simple.plotters.LabelSize attribute), 248
 labelprops (psy_simple.plotters.LabelWeight attribute), 249
 labelprops (psy_simple.plotters.LinePlotter attribute), 264
 labelprops (psy_simple.plotters.Simple2DPlotter attribute), 314
 labelprops (psy_simple.plotters.SimplePlotterBase attribute), 335
 labelprops (psy_simple.plotters.SimpleVectorPlotter attribute), 359
 labelprops (psy_simple.plotters.ViolinPlotter attribute), 401
 labelprops (psy_simple.plotters.XYTickPlotter attribute), 424
 LabelSize (class in psy_simple.plotters), 247
 labelsizes (psy_simple.plotters.BarPlotter attribute), 84
 labelsizes (psy_simple.plotters.CombinedSimplePlotter attribute), 160
 labelsizes (psy_simple.plotters.DensityPlotter attribute), 197
 labelsizes (psy_simple.plotters.FldmeanPlotter attribute), 225
 labelsizes (psy_simple.plotters.LabelProps attribute), 247
 labelsizes (psy_simple.plotters.LinePlotter attribute), 264
 labelsizes (psy_simple.plotters.Simple2DPlotter attribute), 314
 labelsizes (psy_simple.plotters.SimplePlotterBase attribute), 335
 labelsizes (psy_simple.plotters.SimpleVectorPlotter attribute), 359
 labelsizes (psy_simple.plotters.ViolinPlotter attribute), 401
 labelsizes (psy_simple.plotters.XYTickPlotter attribute), 424
 LabelWeight (class in psy_simple.plotters), 248
 labelweight (psy_simple.plotters.BarPlotter attribute), 84
 labelweight (psy_simple.plotters.CombinedSimplePlotter attribute), 160
 labelweight (psy_simple.plotters.DensityPlotter attribute), 197

- labelweight (psy_simple.plotters.FldmeanPlotter attribute), 225
 - labelweight (psy_simple.plotters.LabelProps attribute), 247
 - labelweight (psy_simple.plotters.LinePlotter attribute), 264
 - labelweight (psy_simple.plotters.Simple2DPlotter attribute), 315
 - labelweight (psy_simple.plotters.SimplePlotterBase attribute), 335
 - labelweight (psy_simple.plotters.SimpleVectorPlotter attribute), 359
 - labelweight (psy_simple.plotters.ViolinPlotter attribute), 401
 - labelweight (psy_simple.plotters.XYTickPlotter attribute), 425
 - Legend (class in psy_simple.plotters), 250
 - legend (psy_simple.plotters.BarPlotter attribute), 92
 - legend (psy_simple.plotters.FldmeanPlotter attribute), 236
 - legend (psy_simple.plotters.LinePlotter attribute), 275
 - legend (psy_simple.plotters.Simple2DPlotter attribute), 303
 - legend (psy_simple.plotters.SimplePlotterBase attribute), 330
 - legend (psy_simple.plotters.SimpleVectorPlotter attribute), 348
 - legend (psy_simple.plotters.ViolinPlotter attribute), 409
 - LegendLabels (class in psy_simple.plotters), 251
 - legendlabels (psy_simple.plotters.BarPlotter attribute), 93
 - legendlabels (psy_simple.plotters.FldmeanPlotter attribute), 236
 - legendlabels (psy_simple.plotters.Legend attribute), 251
 - legendlabels (psy_simple.plotters.LinePlotter attribute), 275
 - legendlabels (psy_simple.plotters.Simple2DPlotter attribute), 303
 - legendlabels (psy_simple.plotters.SimplePlotterBase attribute), 330
 - legendlabels (psy_simple.plotters.SimpleVectorPlotter attribute), 348
 - legendlabels (psy_simple.plotters.ViolinPlotter attribute), 409
 - levels (psy_simple.plotters.Cbar attribute), 131
 - levels (psy_simple.plotters.CombinedSimplePlotter attribute), 153
 - levels (psy_simple.plotters.DensityPlotter attribute), 192
 - levels (psy_simple.plotters.Plot2D attribute), 285
 - levels (psy_simple.plotters.Simple2DPlotter attribute), 303
 - levels (psy_simple.plotters.SimplePlot2D attribute), 326
 - levels (psy_simple.plotters.VectorCbar attribute), 387
 - LimitBase (class in psy_simple.plotters), 253
 - LineColors (class in psy_simple.plotters), 254
 - LinePlot (class in psy_simple.plotters), 255
 - lineplot (psypilot.project.plot attribute), 4
 - LinePlotter (class in psy_simple.plotters), 256
 - LineWidth (class in psy_simple.plotters), 277
 - linewidth (psy_simple.plotters.BaseVectorPlotter attribute), 114
 - linewidth (psy_simple.plotters.CombinedBase attribute), 137
 - linewidth (psy_simple.plotters.CombinedSimplePlotter attribute), 176
 - linewidth (psy_simple.plotters.CombinedVectorPlot attribute), 178
 - linewidth (psy_simple.plotters.FldmeanPlotter attribute), 237
 - linewidth (psy_simple.plotters.LinePlotter attribute), 260
 - linewidth (psy_simple.plotters.SimpleVectorPlot attribute), 346
 - linewidth (psy_simple.plotters.SimpleVectorPlotter attribute), 369
 - linewidth (psy_simple.plotters.VectorPlot attribute), 391
 - LineWidthValidator (class in psy_simple.plugin), 448
- ## M
- make_plot() (psy_simple.plotters.BarPlot method), 74
 - make_plot() (psy_simple.plotters.ErrorPlot method), 216
 - make_plot() (psy_simple.plotters.LinePlot method), 255
 - make_plot() (psy_simple.plotters.Plot2D method), 285
 - make_plot() (psy_simple.plotters.VectorPlot method), 391
 - make_plot() (psy_simple.plotters.ViolinPlot method), 392
 - mappable (psy_simple.plotters.Plot2D attribute), 285
 - mappable (psy_simple.plotters.VectorPlot attribute), 391
 - Marker (class in psy_simple.plotters), 278
 - marker (psy_simple.plotters.FldmeanPlotter attribute), 237
 - marker (psy_simple.plotters.Legend attribute), 251
 - marker (psy_simple.plotters.LinePlot attribute), 255
 - marker (psy_simple.plotters.LinePlotter attribute), 260
 - MarkerSize (class in psy_simple.plotters), 279
 - markersize (psy_simple.plotters.FldmeanPlotter attribute), 237
 - markersize (psy_simple.plotters.LinePlotter attribute), 260
 - mask_func() (psy_simple.base.MaskBetween method), 41
 - mask_func() (psy_simple.base.MaskGeq method), 41
 - mask_func() (psy_simple.base.MaskGreater method), 42
 - mask_func() (psy_simple.base.MaskLeq method), 43
 - mask_func() (psy_simple.base.MaskLess method), 44
 - mask_func() (psy_simple.base.ValueMaskBase method), 56
 - MaskBetween (class in psy_simple.base), 40
 - maskbetween (psy_simple.base.BasePlotter attribute), 32

- maskbetween (psy_simple.plotters.BarPlotter attribute), 81
- maskbetween (psy_simple.plotters.CombinedBase attribute), 143
- maskbetween (psy_simple.plotters.CombinedSimplePlotter attribute), 156
- maskbetween (psy_simple.plotters.DensityPlotter attribute), 193
- maskbetween (psy_simple.plotters.FldmeanPlotter attribute), 221
- maskbetween (psy_simple.plotters.LinePlotter attribute), 261
- maskbetween (psy_simple.plotters.ScalarCombinedBase attribute), 290
- maskbetween (psy_simple.plotters.Simple2DPlotter attribute), 310
- maskbetween (psy_simple.plotters.SimplePlotterBase attribute), 332
- maskbetween (psy_simple.plotters.SimpleVectorPlotter attribute), 355
- maskbetween (psy_simple.plotters.ViolinPlotter attribute), 398
- MaskGeq (class in psy_simple.base), 41
- maskgeq (psy_simple.base.BasePlotter attribute), 32
- maskgeq (psy_simple.plotters.BarPlotter attribute), 81
- maskgeq (psy_simple.plotters.CombinedBase attribute), 143
- maskgeq (psy_simple.plotters.CombinedSimplePlotter attribute), 156
- maskgeq (psy_simple.plotters.DensityPlotter attribute), 193
- maskgeq (psy_simple.plotters.FldmeanPlotter attribute), 221
- maskgeq (psy_simple.plotters.LinePlotter attribute), 261
- maskgeq (psy_simple.plotters.ScalarCombinedBase attribute), 290
- maskgeq (psy_simple.plotters.Simple2DPlotter attribute), 311
- maskgeq (psy_simple.plotters.SimplePlotterBase attribute), 333
- maskgeq (psy_simple.plotters.SimpleVectorPlotter attribute), 355
- maskgeq (psy_simple.plotters.ViolinPlotter attribute), 399
- MaskGreater (class in psy_simple.base), 41
- maskgreater (psy_simple.base.BasePlotter attribute), 32
- maskgreater (psy_simple.plotters.BarPlotter attribute), 81
- maskgreater (psy_simple.plotters.CombinedBase attribute), 144
- maskgreater (psy_simple.plotters.CombinedSimplePlotter attribute), 156
- maskgreater (psy_simple.plotters.DensityPlotter attribute), 194
- maskgreater (psy_simple.plotters.FldmeanPlotter attribute), 221
- maskgreater (psy_simple.plotters.LinePlotter attribute), 261
- maskgreater (psy_simple.plotters.ScalarCombinedBase attribute), 291
- maskgreater (psy_simple.plotters.Simple2DPlotter attribute), 311
- maskgreater (psy_simple.plotters.SimplePlotterBase attribute), 333
- maskgreater (psy_simple.plotters.SimpleVectorPlotter attribute), 355
- maskgreater (psy_simple.plotters.ViolinPlotter attribute), 399
- MaskLeq (class in psy_simple.base), 42
- maskleq (psy_simple.base.BasePlotter attribute), 32
- maskleq (psy_simple.plotters.BarPlotter attribute), 81
- maskleq (psy_simple.plotters.CombinedBase attribute), 144
- maskleq (psy_simple.plotters.CombinedSimplePlotter attribute), 157
- maskleq (psy_simple.plotters.DensityPlotter attribute), 194
- maskleq (psy_simple.plotters.FldmeanPlotter attribute), 222
- maskleq (psy_simple.plotters.LinePlotter attribute), 262
- maskleq (psy_simple.plotters.ScalarCombinedBase attribute), 291
- maskleq (psy_simple.plotters.Simple2DPlotter attribute), 311
- maskleq (psy_simple.plotters.SimplePlotterBase attribute), 333
- maskleq (psy_simple.plotters.SimpleVectorPlotter attribute), 355
- maskleq (psy_simple.plotters.ViolinPlotter attribute), 399
- MaskLess (class in psy_simple.base), 43
- maskless (psy_simple.base.BasePlotter attribute), 33
- maskless (psy_simple.plotters.BarPlotter attribute), 82
- maskless (psy_simple.plotters.CombinedBase attribute), 144
- maskless (psy_simple.plotters.CombinedSimplePlotter attribute), 157
- maskless (psy_simple.plotters.DensityPlotter attribute), 194
- maskless (psy_simple.plotters.FldmeanPlotter attribute), 222
- maskless (psy_simple.plotters.LinePlotter attribute), 262
- maskless (psy_simple.plotters.ScalarCombinedBase attribute), 291
- maskless (psy_simple.plotters.Simple2DPlotter attribute), 311
- maskless (psy_simple.plotters.SimplePlotterBase attribute), 333
- maskless (psy_simple.plotters.SimpleVectorPlotter attribute), 356

maskless (psy_simple.plotters.ViolinPlotter attribute), 399
 mean (psy_simple.plotters.ErrorCalculator attribute), 215
 mean (psy_simple.plotters.FldmeanPlotter attribute), 222
 MeanCalculator (class in psy_simple.plotters), 280
 miss_color (psy_simple.plotters.CombinedSimplePlotter attribute), 153
 miss_color (psy_simple.plotters.DensityPlotter attribute), 193
 miss_color (psy_simple.plotters.Simple2DBase attribute), 294
 miss_color (psy_simple.plotters.Simple2DPlotter attribute), 310
 MissColor (class in psy_simple.plotters), 281

N

name (psy_simple.base.Figtitle attribute), 40
 name (psy_simple.base.MaskBetween attribute), 41
 name (psy_simple.base.MaskGeq attribute), 41
 name (psy_simple.base.MaskGreater attribute), 42
 name (psy_simple.base.MaskLeq attribute), 43
 name (psy_simple.base.MaskLess attribute), 44
 name (psy_simple.base.Text attribute), 45
 name (psy_simple.base.Tight attribute), 48
 name (psy_simple.base.Title attribute), 50
 name (psy_simple.plotters.AlternativeXCoord attribute), 66
 name (psy_simple.plotters.ArrowSize attribute), 69
 name (psy_simple.plotters.ArrowStyle attribute), 70
 name (psy_simple.plotters.AxisColor attribute), 72
 name (psy_simple.plotters.BarPlot attribute), 74
 name (psy_simple.plotters.Bounds attribute), 120
 name (psy_simple.plotters.Cbar attribute), 131
 name (psy_simple.plotters.CbarSpacing attribute), 134
 name (psy_simple.plotters.CLabel attribute), 122
 name (psy_simple.plotters.CMap attribute), 124
 name (psy_simple.plotters.ContourLevels attribute), 179
 name (psy_simple.plotters.CTickLabels attribute), 124
 name (psy_simple.plotters.CTickProps attribute), 125
 name (psy_simple.plotters.CTicks attribute), 129
 name (psy_simple.plotters.CTickSize attribute), 127
 name (psy_simple.plotters.CTickWeight attribute), 128
 name (psy_simple.plotters.DataGrid attribute), 180
 name (psy_simple.plotters.DataPrecision attribute), 182
 name (psy_simple.plotters.Density attribute), 183
 name (psy_simple.plotters.ErrorAlpha attribute), 213
 name (psy_simple.plotters.ErrorCalculator attribute), 215
 name (psy_simple.plotters.ErrorPlot attribute), 216
 name (psy_simple.plotters.Extend attribute), 217
 name (psy_simple.plotters.Grid attribute), 240
 name (psy_simple.plotters.Hist2DXRange attribute), 242
 name (psy_simple.plotters.Hist2DYRange attribute), 243
 name (psy_simple.plotters.HistBins attribute), 244
 name (psy_simple.plotters.LabelProps attribute), 247

name (psy_simple.plotters.LabelSize attribute), 248
 name (psy_simple.plotters.LabelWeight attribute), 250
 name (psy_simple.plotters.Legend attribute), 251
 name (psy_simple.plotters.LegendLabels attribute), 252
 name (psy_simple.plotters.LineColors attribute), 254
 name (psy_simple.plotters.LinePlot attribute), 256
 name (psy_simple.plotters.MeanCalculator attribute), 281
 name (psy_simple.plotters.MissColor attribute), 282
 name (psy_simple.plotters.NormedHist2D attribute), 283
 name (psy_simple.plotters.Plot2D attribute), 285
 name (psy_simple.plotters.PointDensity attribute), 287
 name (psy_simple.plotters.SymmetricLimits attribute), 371
 name (psy_simple.plotters.TickSize attribute), 375
 name (psy_simple.plotters.TickWeight attribute), 376
 name (psy_simple.plotters.Transpose attribute), 381
 name (psy_simple.plotters.VectorColor attribute), 388
 name (psy_simple.plotters.VectorLineWidth attribute), 389
 name (psy_simple.plotters.VectorPlot attribute), 391
 name (psy_simple.plotters.ViolinPlot attribute), 392
 name (psy_simple.plotters.Xlabel attribute), 434
 name (psy_simple.plotters.Xlim attribute), 435
 name (psy_simple.plotters.XRotation attribute), 418
 name (psy_simple.plotters.XTickLabels attribute), 418
 name (psy_simple.plotters.XTickProps attribute), 420
 name (psy_simple.plotters.XTicks attribute), 421
 name (psy_simple.plotters.Ylabel attribute), 443
 name (psy_simple.plotters.Ylim attribute), 444
 name (psy_simple.plotters.YRotation attribute), 437
 name (psy_simple.plotters.YTickLabels attribute), 438
 name (psy_simple.plotters.YTickProps attribute), 439
 name (psy_simple.plotters.YTicks attribute), 440
 normed (psy_simple.plotters.DensityPlotter attribute), 188
 normed (psy_simple.plotters.PointDensity attribute), 287
 NormedHist2D (class in psy_simple.plotters), 282

O

original_position (psy_simple.plotters.Cbar attribute), 131

P

parents (psy_simple.plotters.LabelSize attribute), 248
 parents (psy_simple.plotters.LabelWeight attribute), 250
 parents (psy_simple.plotters.VectorBounds attribute), 384
 patch_prior_1_0() (in module psy_simple.plugin), 448
 patches (in module psy_simple.plugin), 448
 plot (psy_simple.plotters.ArrowSize attribute), 69
 plot (psy_simple.plotters.ArrowStyle attribute), 70
 plot (psy_simple.plotters.BarPlotter attribute), 77
 plot (psy_simple.plotters.BarXlim attribute), 98
 plot (psy_simple.plotters.BarXTickLabels attribute), 96
 plot (psy_simple.plotters.BarYlim attribute), 101

- plot (psy_simple.plotters.BarYTickLabels attribute), 99
- plot (psy_simple.plotters.Base2D attribute), 108
- plot (psy_simple.plotters.Cbar attribute), 131
- plot (psy_simple.plotters.CbarOptions attribute), 133
- plot (psy_simple.plotters.CLabel attribute), 122
- plot (psy_simple.plotters.CombinedSimplePlotter attribute), 149
- plot (psy_simple.plotters.CTickLabels attribute), 124
- plot (psy_simple.plotters.CTickProps attribute), 126
- plot (psy_simple.plotters.CTicks attribute), 129
- plot (psy_simple.plotters.CTickSize attribute), 127
- plot (psy_simple.plotters.CTickWeight attribute), 128
- plot (psy_simple.plotters.Density attribute), 183
- plot (psy_simple.plotters.DensityPlotter attribute), 211
- plot (psy_simple.plotters.DtTicksBase attribute), 212
- plot (psy_simple.plotters.ErrorPlot attribute), 216
- plot (psy_simple.plotters.Extend attribute), 217
- plot (psy_simple.plotters.FldmeanPlotter attribute), 239
- plot (psy_simple.plotters.Hist2DXRange attribute), 242
- plot (psy_simple.plotters.Hist2DYRange attribute), 243
- plot (psy_simple.plotters.Legend attribute), 251
- plot (psy_simple.plotters.LimitBase attribute), 253
- plot (psy_simple.plotters.LinePlotter attribute), 260
- plot (psy_simple.plotters.LineWidth attribute), 278
- plot (psy_simple.plotters.MarkerSize attribute), 279
- plot (psy_simple.plotters.MissColor attribute), 282
- plot (psy_simple.plotters.Simple2DPlotter attribute), 304
- plot (psy_simple.plotters.SimpleVectorPlotter attribute), 348
- plot (psy_simple.plotters.TicksBase attribute), 378
- plot (psy_simple.plotters.VCLabel attribute), 382
- plot (psy_simple.plotters.VectorCalculator attribute), 385
- plot (psy_simple.plotters.VectorCbar attribute), 387
- plot (psy_simple.plotters.VectorColor attribute), 388
- plot (psy_simple.plotters.VectorCTicks attribute), 385
- plot (psy_simple.plotters.VectorLineWidth attribute), 389
- plot (psy_simple.plotters.ViolinPlotter attribute), 394
- plot (psy_simple.plotters.ViolinXlim attribute), 414
- plot (psy_simple.plotters.ViolinXTicks attribute), 413
- plot (psy_simple.plotters.ViolinYlim attribute), 416
- plot (psy_simple.plotters.ViolinYTicks attribute), 416
- plot (psy_simple.plotters.Xlim attribute), 435
- plot (psy_simple.plotters.Xlim2D attribute), 436
- plot (psy_simple.plotters.XTicks attribute), 421
- plot (psy_simple.plotters.XTicks2D attribute), 423
- plot (psy_simple.plotters.Ylim attribute), 444
- plot (psy_simple.plotters.Ylim2D attribute), 445
- plot (psy_simple.plotters.YTicks attribute), 440
- plot (psy_simple.plotters.YTicks2D attribute), 441
- Plot2D (class in psy_simple.plotters), 284
- plot2d (psyplot.project.plot attribute), 7
- plot_arr() (psy_simple.plotters.LinePlot method), 256
- plot_fill() (psy_simple.plotters.ErrorPlot method), 216
- plot_fmt (psy_simple.plotters.BarPlot attribute), 74
- plot_fmt (psy_simple.plotters.ErrorPlot attribute), 216
- plot_fmt (psy_simple.plotters.LinePlot attribute), 256
- plot_fmt (psy_simple.plotters.Plot2D attribute), 285
- plot_fmt (psy_simple.plotters.VectorPlot attribute), 391
- plot_fmt (psy_simple.plotters.ViolinPlot attribute), 392
- PointDensity (class in psy_simple.plotters), 286
- post (psy_simple.base.BasePlotter attribute), 37
- post (psy_simple.base.TitlesPlotter attribute), 54
- post (psy_simple.plotters.BarPlotter attribute), 86
- post (psy_simple.plotters.Base2D attribute), 108
- post (psy_simple.plotters.BaseVectorPlotter attribute), 117
- post (psy_simple.plotters.CombinedBase attribute), 144
- post (psy_simple.plotters.CombinedSimplePlotter attribute), 165
- post (psy_simple.plotters.DensityPlotter attribute), 201
- post (psy_simple.plotters.FldmeanPlotter attribute), 228
- post (psy_simple.plotters.LinePlotter attribute), 268
- post (psy_simple.plotters.ScalarCombinedBase attribute), 291
- post (psy_simple.plotters.Simple2DBase attribute), 299
- post (psy_simple.plotters.Simple2DPlotter attribute), 318
- post (psy_simple.plotters.SimplePlotterBase attribute), 339
- post (psy_simple.plotters.SimpleVectorPlotter attribute), 363
- post (psy_simple.plotters.ViolinPlotter attribute), 405
- post (psy_simple.plotters.XYTickPlotter attribute), 431
- post_timing (psy_simple.base.BasePlotter attribute), 38
- post_timing (psy_simple.base.TitlesPlotter attribute), 55
- post_timing (psy_simple.plotters.BarPlotter attribute), 87
- post_timing (psy_simple.plotters.Base2D attribute), 109
- post_timing (psy_simple.plotters.BaseVectorPlotter attribute), 118
- post_timing (psy_simple.plotters.CombinedBase attribute), 145
- post_timing (psy_simple.plotters.CombinedSimplePlotter attribute), 166
- post_timing (psy_simple.plotters.DensityPlotter attribute), 202
- post_timing (psy_simple.plotters.FldmeanPlotter attribute), 229
- post_timing (psy_simple.plotters.LinePlotter attribute), 268
- post_timing (psy_simple.plotters.ScalarCombinedBase attribute), 292
- post_timing (psy_simple.plotters.Simple2DBase attribute), 300
- post_timing (psy_simple.plotters.Simple2DPlotter attribute), 319
- post_timing (psy_simple.plotters.SimplePlotterBase attribute), 340
- post_timing (psy_simple.plotters.SimpleVectorPlotter attribute), 364

post_timing (psy_simple.plotters.ViolinPlotter attribute), 406

post_timing (psy_simple.plotters.XYTickPlotter attribute), 432

precision (psy_simple.plotters.DensityPlotter attribute), 188

precision (psy_simple.plotters.HistBins attribute), 244

precision (psy_simple.plotters.PointDensity attribute), 287

priority (psy_simple.base.ValueMaskBase attribute), 56

priority (psy_simple.plotters.AlternativeXCoord attribute), 66

priority (psy_simple.plotters.ArrowSize attribute), 69

priority (psy_simple.plotters.ArrowStyle attribute), 70

priority (psy_simple.plotters.BarAlpha attribute), 73

priority (psy_simple.plotters.BarPlot attribute), 74

priority (psy_simple.plotters.BarWidths attribute), 95

priority (psy_simple.plotters.Bounds attribute), 120

priority (psy_simple.plotters.Cbar attribute), 131

priority (psy_simple.plotters.CMap attribute), 124

priority (psy_simple.plotters.ContourLevels attribute), 179

priority (psy_simple.plotters.DataPrecision attribute), 182

priority (psy_simple.plotters.Density attribute), 184

priority (psy_simple.plotters.ErrorAlpha attribute), 213

priority (psy_simple.plotters.ErrorCalculator attribute), 215

priority (psy_simple.plotters.ErrorPlot attribute), 216

priority (psy_simple.plotters.Hist2DXRange attribute), 242

priority (psy_simple.plotters.HistBins attribute), 244

priority (psy_simple.plotters.LineColors attribute), 254

priority (psy_simple.plotters.LinePlot attribute), 256

priority (psy_simple.plotters.LineWidth attribute), 278

priority (psy_simple.plotters.Marker attribute), 279

priority (psy_simple.plotters.MarkerSize attribute), 280

priority (psy_simple.plotters.MeanCalculator attribute), 281

priority (psy_simple.plotters.MissColor attribute), 282

priority (psy_simple.plotters.NormedHist2D attribute), 283

priority (psy_simple.plotters.Plot2D attribute), 285

priority (psy_simple.plotters.PointDensity attribute), 287

priority (psy_simple.plotters.Transpose attribute), 381

priority (psy_simple.plotters.VectorCalculator attribute), 385

priority (psy_simple.plotters.VectorCbar attribute), 387

priority (psy_simple.plotters.VectorPlot attribute), 391

priority (psy_simple.plotters.ViolinPlot attribute), 392

psy_simple (module), 30

psy_simple.base (module), 30

psy_simple.colors (module), 58

psy_simple.plotters (module), 62

psy_simple.plugin (module), 446

psy_simple.version (module), 450

R

rc (psy_simple.base.TextBase attribute), 47

rcParams (in module psy_simple.plugin), 448

remove() (psy_simple.base.Text method), 45

remove() (psy_simple.plotters.BarPlot method), 74

remove() (psy_simple.plotters.Cbar method), 131

remove() (psy_simple.plotters.DataGrid method), 180

remove() (psy_simple.plotters.Density method), 184

remove() (psy_simple.plotters.ErrorPlot method), 216

remove() (psy_simple.plotters.Legend method), 251

remove() (psy_simple.plotters.LinePlot method), 256

remove() (psy_simple.plotters.MissColor method), 282

remove() (psy_simple.plotters.Plot2D method), 285

remove() (psy_simple.plotters.VectorPlot method), 391

remove() (psy_simple.plotters.ViolinPlot method), 392

replace() (psy_simple.base.TextBase method), 47

replace_coord() (psy_simple.plotters.AlternativeXCoord method), 66

requires_replot (psy_simple.plotters.ErrorCalculator attribute), 215

requires_replot (psy_simple.plotters.MeanCalculator attribute), 281

round_to_05() (in module psy_simple.plotters), 446

S

ScalarCombinedBase (class in psy_simple.plotters), 287

set_default_formatters() (psy_simple.plotters.CTickLabels method), 124

set_default_formatters() (psy_simple.plotters.TickLabels method), 372

set_default_formatters() (psy_simple.plotters.TickLabelsBase method), 373

set_default_locators() (psy_simple.plotters.CTicks method), 129

set_default_locators() (psy_simple.plotters.TicksBase method), 378

set_formatter() (psy_simple.plotters.CTickLabels method), 124

set_formatter() (psy_simple.plotters.TickLabels method), 372

set_formatter() (psy_simple.plotters.TickLabelsBase method), 373

set_limit() (psy_simple.plotters.Hist2DXRange method), 242

set_limit() (psy_simple.plotters.LimitBase method), 253

set_limit() (psy_simple.plotters.Xlim method), 435

set_limit() (psy_simple.plotters.Ylim method), 444

set_locator() (psy_simple.plotters.TicksBase method), 378

set_stringformatter() (psy_simple.plotters.BarXTickLabels method), 96

- [set_stringformatter\(\) \(psy_simple.plotters.BarYTickLabels method\), 99](#)
[set_stringformatter\(\) \(psy_simple.plotters.TickLabelsBase method\), 373](#)
[set_ticklabels\(\) \(psy_simple.plotters.TickLabelsBase method\), 373](#)
[set_ticks\(\) \(psy_simple.plotters.CTicks method\), 129](#)
[set_ticks\(\) \(psy_simple.plotters.TicksBase method\), 378](#)
[set_value\(\) \(psy_simple.base.Text method\), 45](#)
[set_value\(\) \(psy_simple.plotters.SimpleVectorPlot method\), 346](#)
[share\(\) \(psy_simple.base.Text method\), 46](#)
[show_colormaps\(\) \(in module psy_simple.colors\), 61](#)
[Simple2DBase \(class in psy_simple.plotters\), 292](#)
[Simple2DPlotter \(class in psy_simple.plotters\), 301](#)
[SimplePlot2D \(class in psy_simple.plotters\), 325](#)
[SimplePlotterBase \(class in psy_simple.plotters\), 326](#)
[SimpleVectorPlot \(class in psy_simple.plotters\), 345](#)
[SimpleVectorPlotter \(class in psy_simple.plotters\), 346](#)
[sym_lims \(psy_simple.plotters.BarPlotter attribute\), 93](#)
[sym_lims \(psy_simple.plotters.BarXlim attribute\), 98](#)
[sym_lims \(psy_simple.plotters.BarYlim attribute\), 101](#)
[sym_lims \(psy_simple.plotters.CombinedSimplePlotter attribute\), 176](#)
[sym_lims \(psy_simple.plotters.DensityPlotter attribute\), 210](#)
[sym_lims \(psy_simple.plotters.FldmeanPlotter attribute\), 237](#)
[sym_lims \(psy_simple.plotters.LinePlotter attribute\), 276](#)
[sym_lims \(psy_simple.plotters.Simple2DPlotter attribute\), 324](#)
[sym_lims \(psy_simple.plotters.SimplePlotterBase attribute\), 331](#)
[sym_lims \(psy_simple.plotters.SimpleVectorPlotter attribute\), 370](#)
[sym_lims \(psy_simple.plotters.ViolinPlotter attribute\), 410](#)
[sym_lims \(psy_simple.plotters.ViolinXlim attribute\), 414](#)
[sym_lims \(psy_simple.plotters.ViolinYlim attribute\), 416](#)
[sym_lims \(psy_simple.plotters.Xlim attribute\), 435](#)
[sym_lims \(psy_simple.plotters.Xlim2D attribute\), 436](#)
[sym_lims \(psy_simple.plotters.Ylim attribute\), 444](#)
[sym_lims \(psy_simple.plotters.Ylim2D attribute\), 445](#)
[SymmetricLimits \(class in psy_simple.plotters\), 370](#)
- T**
- [Text \(class in psy_simple.base\), 44](#)
[text \(psy_simple.base.BasePlotter attribute\), 35](#)
[text \(psy_simple.base.TitlesPlotter attribute\), 52](#)
[text \(psy_simple.plotters.BarPlotter attribute\), 84](#)
[text \(psy_simple.plotters.CombinedSimplePlotter attribute\), 161](#)
[text \(psy_simple.plotters.DensityPlotter attribute\), 198](#)
[text \(psy_simple.plotters.FldmeanPlotter attribute\), 225](#)
[text \(psy_simple.plotters.LinePlotter attribute\), 264](#)
[text \(psy_simple.plotters.Simple2DPlotter attribute\), 315](#)
[text \(psy_simple.plotters.SimplePlotterBase attribute\), 336](#)
[text \(psy_simple.plotters.SimpleVectorPlotter attribute\), 359](#)
[text \(psy_simple.plotters.ViolinPlotter attribute\), 402](#)
[TextBase \(class in psy_simple.base\), 46](#)
[TickLabels \(class in psy_simple.plotters\), 372](#)
[TickLabelsBase \(class in psy_simple.plotters\), 372](#)
[TickPropsBase \(class in psy_simple.plotters\), 373](#)
[TicksBase \(class in psy_simple.plotters\), 377](#)
[TickSize \(class in psy_simple.plotters\), 374](#)
[ticksize \(psy_simple.plotters.BarPlotter attribute\), 94](#)
[ticksize \(psy_simple.plotters.CombinedSimplePlotter attribute\), 176](#)
[ticksize \(psy_simple.plotters.DensityPlotter attribute\), 210](#)
[ticksize \(psy_simple.plotters.FldmeanPlotter attribute\), 238](#)
[ticksize \(psy_simple.plotters.LinePlotter attribute\), 276](#)
[ticksize \(psy_simple.plotters.Simple2DPlotter attribute\), 324](#)
[ticksize \(psy_simple.plotters.SimplePlotterBase attribute\), 344](#)
[ticksize \(psy_simple.plotters.SimpleVectorPlotter attribute\), 370](#)
[ticksize \(psy_simple.plotters.ViolinPlotter attribute\), 410](#)
[ticksize \(psy_simple.plotters.XYTickPlotter attribute\), 425](#)
[TickSizeBase \(class in psy_simple.plotters\), 375](#)
[TicksManager \(class in psy_simple.plotters\), 378](#)
[TicksManagerBase \(class in psy_simple.plotters\), 379](#)
[TicksOptions \(class in psy_simple.plotters\), 380](#)
[TicksValidator \(class in psy_simple.plugin\), 448](#)
[TickWeight \(class in psy_simple.plotters\), 375](#)
[tickweight \(psy_simple.plotters.BarPlotter attribute\), 94](#)
[tickweight \(psy_simple.plotters.CombinedSimplePlotter attribute\), 177](#)
[tickweight \(psy_simple.plotters.DensityPlotter attribute\), 211](#)
[tickweight \(psy_simple.plotters.FldmeanPlotter attribute\), 238](#)
[tickweight \(psy_simple.plotters.LinePlotter attribute\), 276](#)
[tickweight \(psy_simple.plotters.Simple2DPlotter attribute\), 325](#)
[tickweight \(psy_simple.plotters.SimplePlotterBase attribute\), 344](#)
[tickweight \(psy_simple.plotters.SimpleVectorPlotter attribute\), 370](#)
[tickweight \(psy_simple.plotters.ViolinPlotter attribute\), 410](#)
[tickweight \(psy_simple.plotters.XYTickPlotter attribute\),](#)

- 425
- TickWeightBase (class in psy_simple.plotters), 376
- Tight (class in psy_simple.base), 47
- tight (psy_simple.base.BasePlotter attribute), 33
- tight (psy_simple.plotters.BarPlotter attribute), 89
- tight (psy_simple.plotters.CombinedSimplePlotter attribute), 167
- tight (psy_simple.plotters.DensityPlotter attribute), 203
- tight (psy_simple.plotters.FldmeanPlotter attribute), 231
- tight (psy_simple.plotters.LinePlotter attribute), 270
- tight (psy_simple.plotters.Simple2DPlotter attribute), 321
- tight (psy_simple.plotters.SimplePlotterBase attribute), 340
- tight (psy_simple.plotters.SimpleVectorPlotter attribute), 365
- tight (psy_simple.plotters.ViolinPlotter attribute), 407
- Title (class in psy_simple.base), 48
- title (psy_simple.base.BasePlotter attribute), 35
- title (psy_simple.base.TitlesPlotter attribute), 53
- title (psy_simple.plotters.BarPlotter attribute), 85
- title (psy_simple.plotters.CombinedSimplePlotter attribute), 161
- title (psy_simple.plotters.DensityPlotter attribute), 199
- title (psy_simple.plotters.FldmeanPlotter attribute), 226
- title (psy_simple.plotters.LinePlotter attribute), 265
- title (psy_simple.plotters.Simple2DPlotter attribute), 316
- title (psy_simple.plotters.SimplePlotterBase attribute), 337
- title (psy_simple.plotters.SimpleVectorPlotter attribute), 360
- title (psy_simple.plotters.ViolinPlotter attribute), 402
- titleprops (psy_simple.base.BasePlotter attribute), 36
- titleprops (psy_simple.base.TitlesPlotter attribute), 54
- titleprops (psy_simple.plotters.BarPlotter attribute), 86
- titleprops (psy_simple.plotters.CombinedSimplePlotter attribute), 162
- titleprops (psy_simple.plotters.DensityPlotter attribute), 199
- titleprops (psy_simple.plotters.FldmeanPlotter attribute), 227
- titleprops (psy_simple.plotters.LinePlotter attribute), 266
- titleprops (psy_simple.plotters.Simple2DPlotter attribute), 316
- titleprops (psy_simple.plotters.SimplePlotterBase attribute), 337
- titleprops (psy_simple.plotters.SimpleVectorPlotter attribute), 361
- titleprops (psy_simple.plotters.ViolinPlotter attribute), 403
- titlesize (psy_simple.base.BasePlotter attribute), 36
- titlesize (psy_simple.base.TitlesPlotter attribute), 54
- titlesize (psy_simple.plotters.BarPlotter attribute), 86
- titlesize (psy_simple.plotters.CombinedSimplePlotter attribute), 162
- titlesize (psy_simple.plotters.DensityPlotter attribute), 199
- titlesize (psy_simple.plotters.FldmeanPlotter attribute), 227
- titlesize (psy_simple.plotters.LinePlotter attribute), 266
- titlesize (psy_simple.plotters.Simple2DPlotter attribute), 317
- titlesize (psy_simple.plotters.SimplePlotterBase attribute), 337
- titlesize (psy_simple.plotters.SimpleVectorPlotter attribute), 361
- titlesize (psy_simple.plotters.ViolinPlotter attribute), 403
- TitlesPlotter (class in psy_simple.base), 50
- titleweight (psy_simple.base.BasePlotter attribute), 37
- titleweight (psy_simple.base.TitlesPlotter attribute), 54
- titleweight (psy_simple.plotters.BarPlotter attribute), 86
- titleweight (psy_simple.plotters.CombinedSimplePlotter attribute), 163
- titleweight (psy_simple.plotters.DensityPlotter attribute), 200
- titleweight (psy_simple.plotters.FldmeanPlotter attribute), 227
- titleweight (psy_simple.plotters.LinePlotter attribute), 266
- titleweight (psy_simple.plotters.Simple2DPlotter attribute), 317
- titleweight (psy_simple.plotters.SimplePlotterBase attribute), 338
- titleweight (psy_simple.plotters.SimpleVectorPlotter attribute), 361
- titleweight (psy_simple.plotters.ViolinPlotter attribute), 404
- transform (psy_simple.base.Text attribute), 46
- transform (psy_simple.plotters.CombinedVectorPlot attribute), 178
- transform (psy_simple.plotters.DataGrid attribute), 180
- transform (psy_simple.plotters.MissColor attribute), 282
- transform (psy_simple.plotters.SimpleVectorPlot attribute), 346
- transform (psy_simple.plotters.VectorPlot attribute), 391
- Transpose (class in psy_simple.plotters), 380
- transpose (psy_simple.plotters.BarPlot attribute), 74
- transpose (psy_simple.plotters.BarPlotter attribute), 89
- transpose (psy_simple.plotters.BarXlabel attribute), 97
- transpose (psy_simple.plotters.BarXlim attribute), 98
- transpose (psy_simple.plotters.BarXTickLabels attribute), 96
- transpose (psy_simple.plotters.BarYlabel attribute), 100
- transpose (psy_simple.plotters.BarYlim attribute), 101
- transpose (psy_simple.plotters.BarYTickLabels attribute), 99
- transpose (psy_simple.plotters.CombinedSimplePlotter attribute), 168
- transpose (psy_simple.plotters.CombinedVectorPlot attribute), 178

- tribute), 178
 - transpose (psy_simple.plotters.DensityPlotter attribute), 204
 - transpose (psy_simple.plotters.DtTicksBase attribute), 212
 - transpose (psy_simple.plotters.ErrorPlot attribute), 216
 - transpose (psy_simple.plotters.FldmeanPlotter attribute), 231
 - transpose (psy_simple.plotters.Hist2DXRange attribute), 242
 - transpose (psy_simple.plotters.Hist2DYRange attribute), 243
 - transpose (psy_simple.plotters.LimitBase attribute), 254
 - transpose (psy_simple.plotters.LinePlot attribute), 256
 - transpose (psy_simple.plotters.LinePlotter attribute), 270
 - transpose (psy_simple.plotters.Simple2DPlotter attribute), 304
 - transpose (psy_simple.plotters.SimplePlot2D attribute), 326
 - transpose (psy_simple.plotters.SimplePlotterBase attribute), 331
 - transpose (psy_simple.plotters.SimpleVectorPlot attribute), 346
 - transpose (psy_simple.plotters.SimpleVectorPlotter attribute), 365
 - transpose (psy_simple.plotters.TickLabels attribute), 372
 - transpose (psy_simple.plotters.TickLabelsBase attribute), 373
 - transpose (psy_simple.plotters.TicksBase attribute), 378
 - transpose (psy_simple.plotters.VectorCalculator attribute), 385
 - transpose (psy_simple.plotters.VectorColor attribute), 388
 - transpose (psy_simple.plotters.VectorLineWidth attribute), 389
 - transpose (psy_simple.plotters.VectorPlot attribute), 391
 - transpose (psy_simple.plotters.ViolinPlot attribute), 392
 - transpose (psy_simple.plotters.ViolinPlotter attribute), 407
 - transpose (psy_simple.plotters.ViolinXlim attribute), 414
 - transpose (psy_simple.plotters.ViolinXTickLabels attribute), 411
 - transpose (psy_simple.plotters.ViolinXTicks attribute), 413
 - transpose (psy_simple.plotters.ViolinYlim attribute), 416
 - transpose (psy_simple.plotters.ViolinYTickLabels attribute), 414
 - transpose (psy_simple.plotters.ViolinYTicks attribute), 416
 - transpose (psy_simple.plotters.Xlabel attribute), 434
 - transpose (psy_simple.plotters.Xlim attribute), 435
 - transpose (psy_simple.plotters.Xlim2D attribute), 436
 - transpose (psy_simple.plotters.XTickLabels attribute), 418
 - transpose (psy_simple.plotters.XTicks attribute), 421
 - transpose (psy_simple.plotters.XTicks2D attribute), 423
 - transpose (psy_simple.plotters.XYTickPlotter attribute), 426
 - transpose (psy_simple.plotters.Ylabel attribute), 443
 - transpose (psy_simple.plotters.Ylim attribute), 444
 - transpose (psy_simple.plotters.Ylim2D attribute), 445
 - transpose (psy_simple.plotters.YTickLabels attribute), 438
 - transpose (psy_simple.plotters.YTicks attribute), 440
 - transpose (psy_simple.plotters.YTicks2D attribute), 441
 - triangles (psy_simple.plotters.DataGrid attribute), 180
 - triangles (psy_simple.plotters.MissColor attribute), 282
 - triangles (psy_simple.plotters.Plot2D attribute), 285
 - triangles (psy_simple.plotters.SimplePlot2D attribute), 326
 - try_and_error() (in module psy_simple.plugin), 448
- ## U
- update() (psy_simple.base.Figtitle method), 40
 - update() (psy_simple.base.Text method), 46
 - update() (psy_simple.base.Tight method), 48
 - update() (psy_simple.base.Title method), 50
 - update() (psy_simple.base.ValueMaskBase method), 56
 - update() (psy_simple.plotters.AlternativeXCoord method), 67
 - update() (psy_simple.plotters.ArrowSize method), 69
 - update() (psy_simple.plotters.ArrowStyle method), 71
 - update() (psy_simple.plotters.AxisColor method), 72
 - update() (psy_simple.plotters.BarAlpha method), 73
 - update() (psy_simple.plotters.BarPlot method), 74
 - update() (psy_simple.plotters.BarWidths method), 95
 - update() (psy_simple.plotters.Bounds method), 120
 - update() (psy_simple.plotters.Cbar method), 131
 - update() (psy_simple.plotters.CbarOptions method), 133
 - update() (psy_simple.plotters.CbarSpacing method), 134
 - update() (psy_simple.plotters.CLabel method), 122
 - update() (psy_simple.plotters.CMap method), 124
 - update() (psy_simple.plotters.CombinedVectorPlot method), 178
 - update() (psy_simple.plotters.ContourLevels method), 179
 - update() (psy_simple.plotters.CTicks method), 129
 - update() (psy_simple.plotters.DataGrid method), 181
 - update() (psy_simple.plotters.DataPrecision method), 182
 - update() (psy_simple.plotters.Density method), 184
 - update() (psy_simple.plotters.DtTicksBase method), 212
 - update() (psy_simple.plotters.ErrorAlpha method), 213
 - update() (psy_simple.plotters.ErrorCalculator method), 215
 - update() (psy_simple.plotters.ErrorPlot method), 216
 - update() (psy_simple.plotters.Extend method), 217
 - update() (psy_simple.plotters.Grid method), 240

- update() (psy_simple.plotters.HistBins method), 244
 - update() (psy_simple.plotters.InterpolateBounds method), 245
 - update() (psy_simple.plotters.LabelOptions method), 246
 - update() (psy_simple.plotters.Legend method), 251
 - update() (psy_simple.plotters.LegendLabels method), 252
 - update() (psy_simple.plotters.LimitBase method), 254
 - update() (psy_simple.plotters.LineColors method), 254
 - update() (psy_simple.plotters.LinePlot method), 256
 - update() (psy_simple.plotters.LineWidth method), 278
 - update() (psy_simple.plotters.Marker method), 279
 - update() (psy_simple.plotters.MarkerSize method), 280
 - update() (psy_simple.plotters.MeanCalculator method), 281
 - update() (psy_simple.plotters.MissColor method), 282
 - update() (psy_simple.plotters.NormedHist2D method), 283
 - update() (psy_simple.plotters.Plot2D method), 285
 - update() (psy_simple.plotters.PointDensity method), 287
 - update() (psy_simple.plotters.SymmetricLimits method), 371
 - update() (psy_simple.plotters.TickLabels method), 372
 - update() (psy_simple.plotters.TicksManager method), 379
 - update() (psy_simple.plotters.TicksOptions method), 380
 - update() (psy_simple.plotters.Transpose method), 381
 - update() (psy_simple.plotters.VectorBounds method), 384
 - update() (psy_simple.plotters.VectorCbar method), 387
 - update() (psy_simple.plotters.VectorColor method), 388
 - update() (psy_simple.plotters.VectorLineWidth method), 389
 - update() (psy_simple.plotters.VectorPlot method), 391
 - update() (psy_simple.plotters.ViolinPlot method), 392
 - update() (psy_simple.plotters.Xlabel method), 434
 - update() (psy_simple.plotters.XRotation method), 418
 - update() (psy_simple.plotters.Ylabel method), 443
 - update() (psy_simple.plotters.YRotation method), 437
 - update_after_plot (psy_simple.plotters.BarXlabel attribute), 97
 - update_after_plot (psy_simple.plotters.BarYlabel attribute), 100
 - update_after_plot (psy_simple.plotters.MissColor attribute), 282
 - update_axis() (psy_simple.plotters.CTickProps method), 126
 - update_axis() (psy_simple.plotters.CTicks method), 129
 - update_axis() (psy_simple.plotters.LabelOptions method), 246
 - update_axis() (psy_simple.plotters.LabelProps method), 247
 - update_axis() (psy_simple.plotters.LabelSize method), 248
 - update_axis() (psy_simple.plotters.LabelWeight method), 250
 - update_axis() (psy_simple.plotters.TickLabelsBase method), 373
 - update_axis() (psy_simple.plotters.TickPropsBase method), 374
 - update_axis() (psy_simple.plotters.TicksBase method), 378
 - update_axis() (psy_simple.plotters.TickSizeBase method), 375
 - update_axis() (psy_simple.plotters.TicksManagerBase method), 380
 - update_axis() (psy_simple.plotters.TickWeightBase method), 377
 - update_axis() (psy_simple.plotters.ViolinXTickLabels method), 411
 - update_axis() (psy_simple.plotters.ViolinYTickLabels method), 414
 - update_colorbar() (psy_simple.plotters.Cbar method), 132
 - use_raw_data (psy_simple.plotters.AlternativeXCoord attribute), 67
 - use_raw_data (psy_simple.plotters.AlternativeXCoordPost attribute), 68
- ## V
- validate_alpha() (in module psy_simple.plugin), 449
 - validate_axiscolor() (in module psy_simple.plugin), 449
 - validate_cbarpos() (in module psy_simple.plugin), 449
 - validate_cmap() (in module psy_simple.plugin), 449
 - validate_cmmaps() (in module psy_simple.plugin), 449
 - validate_dataarray() (in module psy_simple.plugin), 449
 - validate_err_calc() (in module psy_simple.plugin), 449
 - validate_float() (in module psy_simple.plugin), 449
 - validate_fontweight() (in module psy_simple.plugin), 449
 - validate_iter() (in module psy_simple.plugin), 450
 - validate_legend() (in module psy_simple.plugin), 450
 - validate_limits() (in module psy_simple.plugin), 450
 - validate_lineplot() (in module psy_simple.plugin), 450
 - validate_marker() (in module psy_simple.plugin), 450
 - validate_none() (in module psy_simple.plugin), 450
 - validate_str() (in module psy_simple.plugin), 450
 - validate_sym_lims() (in module psy_simple.plugin), 450
 - validate_text() (in module psy_simple.plugin), 450
 - validate_ticklabels() (in module psy_simple.plugin), 450
 - ValidateList (class in psy_simple.plugin), 448
 - value2pickle (psy_simple.plotters.AxisColor attribute), 72
 - value2pickle (psy_simple.plotters.LineColors attribute), 255
 - value2share (psy_simple.plotters.Bounds attribute), 120
 - value2share (psy_simple.plotters.Cbar attribute), 132
 - value2share (psy_simple.plotters.LimitBase attribute), 254

value2share (psy_simple.plotters.LineColors attribute), 255

ValueMaskBase (class in psy_simple.base), 55

vbounds (psy_simple.plotters.CombinedBase attribute), 137

vbounds (psy_simple.plotters.CombinedSimplePlotter attribute), 154

vcbar (psy_simple.plotters.CombinedBase attribute), 138

vcbar (psy_simple.plotters.CombinedSimplePlotter attribute), 155

vcbarspacing (psy_simple.plotters.CombinedBase attribute), 138

vcbarspacing (psy_simple.plotters.CombinedSimplePlotter attribute), 155

VCLabel (class in psy_simple.plotters), 381

vclabel (psy_simple.plotters.CombinedBase attribute), 139

vclabel (psy_simple.plotters.CombinedSimplePlotter attribute), 163

vclabelprops (psy_simple.plotters.CombinedBase attribute), 139

vclabelprops (psy_simple.plotters.CombinedSimplePlotter attribute), 163

vclabelsize (psy_simple.plotters.CombinedBase attribute), 139

vclabelsize (psy_simple.plotters.CombinedSimplePlotter attribute), 164

vclabelweight (psy_simple.plotters.CombinedBase attribute), 140

vclabelweight (psy_simple.plotters.CombinedSimplePlotter attribute), 164

vcmap (psy_simple.plotters.CombinedBase attribute), 140

vcmap (psy_simple.plotters.CombinedSimplePlotter attribute), 155

vcticklabels (psy_simple.plotters.CombinedBase attribute), 140

vcticklabels (psy_simple.plotters.CombinedSimplePlotter attribute), 170

vctickprops (psy_simple.plotters.CombinedBase attribute), 140

vctickprops (psy_simple.plotters.CombinedSimplePlotter attribute), 155

vcticks (psy_simple.plotters.CombinedBase attribute), 141

vcticks (psy_simple.plotters.CombinedSimplePlotter attribute), 170

vcticksiz (psy_simple.plotters.CombinedBase attribute), 141

vcticksiz (psy_simple.plotters.CombinedSimplePlotter attribute), 156

vctickweight (psy_simple.plotters.CombinedBase attribute), 142

vctickweight (psy_simple.plotters.CombinedSimplePlotter attribute), 156

vector (psyplot.project.plot attribute), 5

VectorBounds (class in psy_simple.plotters), 382

VectorCalculator (class in psy_simple.plotters), 385

VectorCbar (class in psy_simple.plotters), 385

VectorColor (class in psy_simple.plotters), 387

VectorCTicks (class in psy_simple.plotters), 384

VectorLineWidth (class in psy_simple.plotters), 388

VectorPlot (class in psy_simple.plotters), 389

ViolinPlot (class in psy_simple.plotters), 391

violinplot (psyplot.project.plot attribute), 6

ViolinPlotter (class in psy_simple.plotters), 392

ViolinXlim (class in psy_simple.plotters), 413

ViolinXTickLabels (class in psy_simple.plotters), 411

ViolinXTicks (class in psy_simple.plotters), 411

ViolinYlim (class in psy_simple.plotters), 416

ViolinYTickLabels (class in psy_simple.plotters), 414

ViolinYTicks (class in psy_simple.plotters), 414

vplot (psy_simple.plotters.CombinedSimplePlotter attribute), 149

W

which (psy_simple.plotters.CbarOptions attribute), 133

widths (psy_simple.plotters.BarPlot attribute), 74

widths (psy_simple.plotters.BarPlotter attribute), 77

widths (psy_simple.plotters.BarXlim attribute), 98

widths (psy_simple.plotters.BarXTickLabels attribute), 96

widths (psy_simple.plotters.BarYlim attribute), 101

X

xbounds (psy_simple.plotters.DataGrid attribute), 181

xbounds (psy_simple.plotters.Plot2D attribute), 286

xbounds (psy_simple.plotters.SimplePlot2D attribute), 326

xcoord (psy_simple.plotters.Plot2D attribute), 286

xcoord (psy_simple.plotters.SimplePlot2D attribute), 326

xcoord (psy_simple.plotters.VectorPlot attribute), 391

Xlabel (class in psy_simple.plotters), 432

xlabel (psy_simple.plotters.BarPlotter attribute), 78

xlabel (psy_simple.plotters.CombinedSimplePlotter attribute), 164

xlabel (psy_simple.plotters.DensityPlotter attribute), 200

xlabel (psy_simple.plotters.FldmeanPlotter attribute), 227

xlabel (psy_simple.plotters.LabelOptions attribute), 246

xlabel (psy_simple.plotters.LabelProps attribute), 247

xlabel (psy_simple.plotters.LabelSize attribute), 248

xlabel (psy_simple.plotters.LabelWeight attribute), 250

xlabel (psy_simple.plotters.LinePlotter attribute), 266

xlabel (psy_simple.plotters.Simple2DPlotter attribute), 317

xlabel (psy_simple.plotters.SimplePlotterBase attribute), 338

- xlabel (psy_simple.plotters.SimpleVectorPlotter attribute), 362
- xlabel (psy_simple.plotters.ViolinPlotter attribute), 404
- xlabel (psy_simple.plotters.XYTickPlotter attribute), 426
- Xlim (class in psy_simple.plotters), 434
- xlim (psy_simple.plotters.BarPlotter attribute), 78
- xlim (psy_simple.plotters.BarYlim attribute), 101
- xlim (psy_simple.plotters.CombinedSimplePlotter attribute), 168
- xlim (psy_simple.plotters.DensityPlotter attribute), 204
- xlim (psy_simple.plotters.FldmeanPlotter attribute), 231
- xlim (psy_simple.plotters.LinePlotter attribute), 270
- xlim (psy_simple.plotters.Simple2DPlotter attribute), 304
- xlim (psy_simple.plotters.SimplePlotterBase attribute), 331
- xlim (psy_simple.plotters.SimpleVectorPlotter attribute), 349
- xlim (psy_simple.plotters.SymmetricLimits attribute), 371
- xlim (psy_simple.plotters.ViolinPlotter attribute), 394
- xlim (psy_simple.plotters.ViolinYlim attribute), 417
- xlim (psy_simple.plotters.Ylim attribute), 444
- xlim (psy_simple.plotters.Ylim2D attribute), 445
- Xlim2D (class in psy_simple.plotters), 435
- xrange (psy_simple.plotters.DataPrecision attribute), 182
- xrange (psy_simple.plotters.DensityPlotter attribute), 188
- xrange (psy_simple.plotters.PointDensity attribute), 287
- XRotation (class in psy_simple.plotters), 417
- xrotation (psy_simple.plotters.BarPlotter attribute), 89
- xrotation (psy_simple.plotters.CombinedSimplePlotter attribute), 171
- xrotation (psy_simple.plotters.DensityPlotter attribute), 206
- xrotation (psy_simple.plotters.FldmeanPlotter attribute), 232
- xrotation (psy_simple.plotters.LinePlotter attribute), 271
- xrotation (psy_simple.plotters.Simple2DPlotter attribute), 322
- xrotation (psy_simple.plotters.SimplePlotterBase attribute), 340
- xrotation (psy_simple.plotters.SimpleVectorPlotter attribute), 366
- xrotation (psy_simple.plotters.ViolinPlotter attribute), 408
- xrotation (psy_simple.plotters.XYTickPlotter attribute), 427
- XTickLabels (class in psy_simple.plotters), 418
- xticklabels (psy_simple.plotters.BarPlotter attribute), 79
- xticklabels (psy_simple.plotters.CombinedSimplePlotter attribute), 171
- xticklabels (psy_simple.plotters.DensityPlotter attribute), 206
- xticklabels (psy_simple.plotters.FldmeanPlotter attribute), 232
- xticklabels (psy_simple.plotters.LinePlotter attribute), 271
- xticklabels (psy_simple.plotters.Simple2DPlotter attribute), 322
- xticklabels (psy_simple.plotters.SimplePlotterBase attribute), 341
- xticklabels (psy_simple.plotters.SimpleVectorPlotter attribute), 367
- xticklabels (psy_simple.plotters.ViolinPlotter attribute), 395
- xticklabels (psy_simple.plotters.XYTickPlotter attribute), 427
- XTickProps (class in psy_simple.plotters), 419
- xtickprops (psy_simple.plotters.BarPlotter attribute), 89
- xtickprops (psy_simple.plotters.CombinedSimplePlotter attribute), 171
- xtickprops (psy_simple.plotters.DensityPlotter attribute), 206
- xtickprops (psy_simple.plotters.FldmeanPlotter attribute), 233
- xtickprops (psy_simple.plotters.LinePlotter attribute), 272
- xtickprops (psy_simple.plotters.Simple2DPlotter attribute), 322
- xtickprops (psy_simple.plotters.SimplePlotterBase attribute), 341
- xtickprops (psy_simple.plotters.SimpleVectorPlotter attribute), 367
- xtickprops (psy_simple.plotters.TickSize attribute), 375
- xtickprops (psy_simple.plotters.TickWeight attribute), 376
- xtickprops (psy_simple.plotters.ViolinPlotter attribute), 408
- xtickprops (psy_simple.plotters.XYTickPlotter attribute), 427
- XTicks (class in psy_simple.plotters), 420
- xticks (psy_simple.plotters.BarPlotter attribute), 90
- xticks (psy_simple.plotters.BarXlim attribute), 98
- xticks (psy_simple.plotters.BarXTickLabels attribute), 96
- xticks (psy_simple.plotters.CombinedSimplePlotter attribute), 172
- xticks (psy_simple.plotters.DensityPlotter attribute), 207
- xticks (psy_simple.plotters.FldmeanPlotter attribute), 233
- xticks (psy_simple.plotters.LinePlotter attribute), 272
- xticks (psy_simple.plotters.Simple2DPlotter attribute), 305
- xticks (psy_simple.plotters.SimplePlotterBase attribute), 341
- xticks (psy_simple.plotters.SimpleVectorPlotter attribute), 349
- xticks (psy_simple.plotters.ViolinPlotter attribute), 395
- xticks (psy_simple.plotters.ViolinXlim attribute), 414
- xticks (psy_simple.plotters.ViolinXTickLabels attribute), 411

xticks (psy_simple.plotters.Xlim attribute), 435
 xticks (psy_simple.plotters.Xlim2D attribute), 436
 xticks (psy_simple.plotters.XTickLabels attribute), 419
 xticks (psy_simple.plotters.XYTickPlotter attribute), 427
 XTicks2D (class in psy_simple.plotters), 421
 XYTickPlotter (class in psy_simple.plotters), 423

Y

ybounds (psy_simple.plotters.DataGrid attribute), 181
 ybounds (psy_simple.plotters.Plot2D attribute), 286
 ybounds (psy_simple.plotters.SimplePlot2D attribute), 326
 ycoord (psy_simple.plotters.Plot2D attribute), 286
 ycoord (psy_simple.plotters.SimplePlot2D attribute), 326
 ycoord (psy_simple.plotters.VectorPlot attribute), 391
 Ylabel (class in psy_simple.plotters), 441
 ylabel (psy_simple.plotters.BarPlotter attribute), 79
 ylabel (psy_simple.plotters.BarXlabel attribute), 97
 ylabel (psy_simple.plotters.CombinedSimplePlotter attribute), 165
 ylabel (psy_simple.plotters.DensityPlotter attribute), 200
 ylabel (psy_simple.plotters.FldmeanPlotter attribute), 228
 ylabel (psy_simple.plotters.LabelOptions attribute), 246
 ylabel (psy_simple.plotters.LabelProps attribute), 247
 ylabel (psy_simple.plotters.LabelSize attribute), 248
 ylabel (psy_simple.plotters.LabelWeight attribute), 250
 ylabel (psy_simple.plotters.LinePlotter attribute), 267
 ylabel (psy_simple.plotters.Simple2DPlotter attribute), 318
 ylabel (psy_simple.plotters.SimplePlotterBase attribute), 338
 ylabel (psy_simple.plotters.SimpleVectorPlotter attribute), 362
 ylabel (psy_simple.plotters.ViolinPlotter attribute), 404
 ylabel (psy_simple.plotters.Xlabel attribute), 434
 ylabel (psy_simple.plotters.XYTickPlotter attribute), 429
 Ylim (class in psy_simple.plotters), 443
 ylim (psy_simple.plotters.BarPlotter attribute), 80
 ylim (psy_simple.plotters.BarXlim attribute), 98
 ylim (psy_simple.plotters.CombinedSimplePlotter attribute), 168
 ylim (psy_simple.plotters.DensityPlotter attribute), 204
 ylim (psy_simple.plotters.FldmeanPlotter attribute), 232
 ylim (psy_simple.plotters.LinePlotter attribute), 271
 ylim (psy_simple.plotters.Simple2DPlotter attribute), 306
 ylim (psy_simple.plotters.SimplePlotterBase attribute), 332
 ylim (psy_simple.plotters.SimpleVectorPlotter attribute), 350
 ylim (psy_simple.plotters.SymmetricLimits attribute), 371
 ylim (psy_simple.plotters.ViolinPlotter attribute), 396
 ylim (psy_simple.plotters.ViolinXlim attribute), 414
 ylim (psy_simple.plotters.Xlim attribute), 435

ylim (psy_simple.plotters.Xlim2D attribute), 436
 Ylim2D (class in psy_simple.plotters), 444
 yrange (psy_simple.plotters.DataPrecision attribute), 182
 yrange (psy_simple.plotters.DensityPlotter attribute), 189
 yrange (psy_simple.plotters.PointDensity attribute), 287
 YRotation (class in psy_simple.plotters), 436
 yrotation (psy_simple.plotters.BarPlotter attribute), 91
 yrotation (psy_simple.plotters.CombinedSimplePlotter attribute), 173
 yrotation (psy_simple.plotters.DensityPlotter attribute), 208
 yrotation (psy_simple.plotters.FldmeanPlotter attribute), 234
 yrotation (psy_simple.plotters.LinePlotter attribute), 273
 yrotation (psy_simple.plotters.Simple2DPlotter attribute), 323
 yrotation (psy_simple.plotters.SimplePlotterBase attribute), 342
 yrotation (psy_simple.plotters.SimpleVectorPlotter attribute), 367
 yrotation (psy_simple.plotters.ViolinPlotter attribute), 408
 yrotation (psy_simple.plotters.XYTickPlotter attribute), 429
 YTickLabels (class in psy_simple.plotters), 437
 yticklabels (psy_simple.plotters.BarPlotter attribute), 80
 yticklabels (psy_simple.plotters.BarXTickLabels attribute), 96
 yticklabels (psy_simple.plotters.CombinedSimplePlotter attribute), 173
 yticklabels (psy_simple.plotters.DensityPlotter attribute), 208
 yticklabels (psy_simple.plotters.FldmeanPlotter attribute), 234
 yticklabels (psy_simple.plotters.LinePlotter attribute), 273
 yticklabels (psy_simple.plotters.Simple2DPlotter attribute), 323
 yticklabels (psy_simple.plotters.SimplePlotterBase attribute), 343
 yticklabels (psy_simple.plotters.SimpleVectorPlotter attribute), 367
 yticklabels (psy_simple.plotters.ViolinPlotter attribute), 397
 yticklabels (psy_simple.plotters.ViolinXTickLabels attribute), 411
 yticklabels (psy_simple.plotters.XRotation attribute), 418
 yticklabels (psy_simple.plotters.XTickLabels attribute), 419
 yticklabels (psy_simple.plotters.XYTickPlotter attribute), 429
 yticklabels (psy_simple.plotters.YRotation attribute), 437
 YTickProps (class in psy_simple.plotters), 438
 ytickprops (psy_simple.plotters.BarPlotter attribute), 91

ytickprops (psy_simple.plotters.CombinedSimplePlotter attribute), [173](#)
ytickprops (psy_simple.plotters.DensityPlotter attribute), [208](#)
ytickprops (psy_simple.plotters.FldmeanPlotter attribute), [235](#)
ytickprops (psy_simple.plotters.LinePlotter attribute), [274](#)
ytickprops (psy_simple.plotters.Simple2DPlotter attribute), [323](#)
ytickprops (psy_simple.plotters.SimplePlotterBase attribute), [343](#)
ytickprops (psy_simple.plotters.SimpleVectorPlotter attribute), [368](#)
ytickprops (psy_simple.plotters.TickSize attribute), [375](#)
ytickprops (psy_simple.plotters.TickWeight attribute), [376](#)
ytickprops (psy_simple.plotters.ViolinPlotter attribute), [408](#)
ytickprops (psy_simple.plotters.XYTickPlotter attribute), [430](#)
YTicks (class in psy_simple.plotters), [439](#)
yticks (psy_simple.plotters.BarPlotter attribute), [91](#)
yticks (psy_simple.plotters.BarYlim attribute), [101](#)
yticks (psy_simple.plotters.BarYTickLabels attribute), [99](#)
yticks (psy_simple.plotters.CombinedSimplePlotter attribute), [174](#)
yticks (psy_simple.plotters.DensityPlotter attribute), [209](#)
yticks (psy_simple.plotters.FldmeanPlotter attribute), [235](#)
yticks (psy_simple.plotters.LinePlotter attribute), [274](#)
yticks (psy_simple.plotters.Simple2DPlotter attribute), [307](#)
yticks (psy_simple.plotters.SimplePlotterBase attribute), [343](#)
yticks (psy_simple.plotters.SimpleVectorPlotter attribute), [351](#)
yticks (psy_simple.plotters.ViolinPlotter attribute), [397](#)
yticks (psy_simple.plotters.ViolinXTicks attribute), [413](#)
yticks (psy_simple.plotters.ViolinYlim attribute), [417](#)
yticks (psy_simple.plotters.ViolinYTickLabels attribute), [414](#)
yticks (psy_simple.plotters.XTicks attribute), [421](#)
yticks (psy_simple.plotters.XTicks2D attribute), [423](#)
yticks (psy_simple.plotters.XYTickPlotter attribute), [430](#)
yticks (psy_simple.plotters.Ylim attribute), [444](#)
yticks (psy_simple.plotters.Ylim2D attribute), [445](#)
yticks (psy_simple.plotters.YTickLabels attribute), [438](#)
YTicks2D (class in psy_simple.plotters), [440](#)