
pyrad library reference for developers

Release 0.0.1

meteoswiss-mdr

Feb 07, 2019

CONTENTS

1	pyrad.flow.flow_aux	3
2	pyrad.flow.flow_control	9
3	pyrad.proc.process_aux	11
4	pyrad.proc.process_calib	15
5	pyrad.proc.process_cosmo	19
6	pyrad.proc.process_Doppler	23
7	pyrad.proc.process_echoclass	27
8	pyrad.proc.process_intercomp	33
9	pyrad.proc.process_monitoring	37
10	pyrad.proc.process_phase	43
11	pyrad.proc.process_retrieve	49
12	pyrad.proc.process_timeseries	55
13	pyrad.proc.process_traj	61
14	pyrad.prod.product_aux	69
15	pyrad.prod.process_product	71
16	pyrad.prod.process_vol_products	73
17	pyrad.prod.process_grid_products	75
18	pyrad.prod.process_timeseries_products	77
19	pyrad.prod.process_monitoring_products	79
20	pyrad.prod.process_intercomp_products	81
21	pyrad.prod.process_product	83
22	pyrad.io.io_aux	85

23	<code>pyrad.io.config</code>	91
24	<code>pyrad.io.read_data_radar</code>	93
25	<code>pyrad.io.read_data_mxpola</code>	101
26	<code>pyrad.io.read_data_cosmo</code>	121
27	<code>pyrad.io.read_data_hzt</code>	125
28	<code>pyrad.io.read_data_sensor</code>	127
29	<code>pyrad.io.read_data_sun</code>	133
30	<code>pyrad.io.read_data_other</code>	135
31	<code>pyrad.io.write_data</code>	141
32	<code>pyrad.io.timeseries</code>	149
33	<code>pyrad.io.trajectory</code>	153
34	<code>pyrad.graph.plots_aux</code>	159
35	<code>pyrad.graph.plots</code>	161
36	<code>pyrad.graph.plots_vol</code>	165
37	<code>pyrad.graph.plots_grid</code>	173
38	<code>pyrad.graph.plot_timeseries</code>	175
39	<code>pyrad.util.radar_utils</code>	179
40	<code>pyrad.util.stat_utils</code>	189
41	Indices and tables	191
	Python Module Index	193
	Index	195

Contents:

PYRAD.FLOW.FLOW_AUX

Auxiliary functions to control the Pyrad data processing flow

<code>_initialize_listener()</code>	initialize the input listener
<code>_user_input_listener(input_queue)</code>	Permanently listens to the keyword input until the user types “Return”
<code>_get_times_and_traj(*args, **kwargs)</code>	wrapper
<code>_initialize_datasets(dataset_levels, cfg[, ...])</code>	Initializes datasets.
<code>_process_datasets(*args, **kwargs)</code>	wrapper
<code>_postprocess_datasets(dataset_levels, cfg, dscfg)</code>	Processes the radar volumes for a particular time stamp.
<code>_wait_for_files(nowtime, datacfg, datatype_list)</code>	Waits for the master file and all files in a volume scan to be present returns the masterfile if the volume scan can be processed.
<code>_get_radars_data(*args, **kwargs)</code>	wrapper
<code>_generate_dataset(*args, **kwargs)</code>	wrapper
<code>_generate_prod(*args, **kwargs)</code>	wrapper
<code>_create_cfg_dict(*args, **kwargs)</code>	wrapper
<code>_create_datacfg_dict(*args, **kwargs)</code>	wrapper
<code>_create_dscfg_dict(*args, **kwargs)</code>	wrapper
<code>_create_prdcfg_dict(*args, **kwargs)</code>	wrapper
<code>_get_datatype_list(*args, **kwargs)</code>	wrapper
<code>_get_datasets_list(*args, **kwargs)</code>	wrapper
<code>_get_masterfile_list(*args, **kwargs)</code>	wrapper
<code>_add_dataset(*args, **kwargs)</code>	wrapper
<code>_warning_format(message, category, filename, ...)</code>	

`pyrad.flow.flow_aux._add_dataset (*args, **kwargs)`
wrapper

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

`pyrad.flow.flow_aux._create_cfg_dict (*args, **kwargs)`
wrapper

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

```
pyrad.flow.flow_aux._create_datacfg_dict(*args, **kwargs)
wrapper
```

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

```
pyrad.flow.flow_aux._create_dscfg_dict(*args, **kwargs)
wrapper
```

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

```
pyrad.flow.flow_aux._create_prdcfg_dict(*args, **kwargs)
wrapper
```

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

```
pyrad.flow.flow_aux._generate_dataset(*args, **kwargs)
wrapper
```

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

```
pyrad.flow.flow_aux._generate_prod(*args, **kwargs)
wrapper
```

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

`pyrad.flow.flow_aux._get_datasets_list(*args, **kwargs)`
wrapper

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

`pyrad.flow.flow_aux._get_datatype_list(*args, **kwargs)`
wrapper

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

`pyrad.flow.flow_aux._get_masterfile_list(*args, **kwargs)`
wrapper

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

`pyrad.flow.flow_aux._get_radars_data(*args, **kwargs)`
wrapper

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

`pyrad.flow.flow_aux._get_times_and_traj(*args, **kwargs)`
wrapper

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

`pyrad.flow.flow_aux._initialize_datasets(dataset_levels, cfg, traj=None, infostr=None)`
Initializes datasets. Creates the data set configuration dictionary

Parameters

dataset_levels [dict] dictionary containing the list of data sets to be generated at each processing level

cfg [dict] processing configuration dictionary

traj [trajectory object] object containing the trajectory

infostr [str] Information string about the actual data processing (e.g. 'RUN57'). This string is added to product files.

Returns

dscfg [dict] dictionary containing the configuration data for each dataset

traj [trajectory object] the modified trajectory object

`pyrad.flow.flow_aux._initialize_listener()`
initialize the input listener

Returns

input_queue [queue object] the queue object where to put the quit signal

`pyrad.flow.flow_aux._postprocess_datasets(dataset_levels, cfg, dscfg, traj=None, infostr=None)`
Processes the radar volumes for a particular time stamp.

Parameters

dataset_levels [dict] dictionary containing the list of data sets to be generated at each processing level

cfg [dict] processing configuration dictionary

dscfg [dict] dictionary containing the configuration data for each dataset

traj [trajectory object] and object containing the trajectory

infostr [str] Information string about the actual data processing (e.g. 'RUN57'). This string is added to product files.

Returns

dscfg [dict] the modified configuration dictionary

traj [trajectory object] the modified trajectory object

`pyrad.flow.flow_aux._process_datasets(*args, **kwargs)`
wrapper

Parameters

args, kwargs [arguments] The arguments of the function

Returns

func [function] The original function if no profiling has to be performed or the function decorated with the memory decorator

`pyrad.flow.flow_aux._user_input_listener(input_queue)`
Permanently listens to the keyword input until the user types "Return"

Parameters

input_queue [queue object] the queue object where to put the quit signal

`pyrad.flow.flow_aux._wait_for_files(nowtime, datacfg, datatype_list, last_processed=None)`
Waits for the master file and all files in a volume scan to be present returns the masterfile if the volume scan can be processed.

Parameters

nowtime [datetime object] the current time

datacfg [dict] dictionary containing the parameters to get the radar data

last_processed [datetime or None] The end time of the previously processed radar volume

Returns

masterfile [str or None] name of the master file. None if the volume was not complete

masterdatatypedescr [str] the description of the master data type

last_processed [datetime] True of all scans found

`pyrad.flow.flow_aux._wait_for_rainbow_datatypes` (*rainbow_files*, *period=30*)

waits until the files for all rainbow data types are present.

Parameters

rainbow_files [list of strings] a list containing the names of all the rainbow files to wait for

period [int] the time it has to wait (s)

Returns

found_all [Boolean] True if all files were present. False otherwise

`pyrad.flow.flow_aux._warning_format` (*message*, *category*, *filename*, *lineno*, *file=None*,
line=None)

`pyrad.flow.flow_aux.profiler` (*level=1*)

Function to be used as decorator for memory debugging. The function will be profiled or not according to its level respect to the global variable PROFILE_LEVEL

Parameters

level [int] profiling level

Returns

func or func wrapper [function] The function or its wrapper for profiling

PYRAD.FLOW.FLOW_CONTROL

functions to control the Pyrad data processing flow

<code>main(cfgfile[, starttime, endtime, ...])</code>	Main flow control.
<code>main_rt(cfgfile_list[, starttime, endtime, ...])</code>	main flow control.

```
pyrad.flow.flow_control.main(cfgfile, starttime=None, endtime=None, trajfile="", trajtype='plane', flashnr=0, infostr="", MULTIPROCESSING_DSET=False, MULTIPROCESSING_PROD=False, PRO-FILE_MULTIPROCESSING=False)
```

Main flow control. Processes radar data off-line over a period of time given either by the user, a trajectory file, or determined by the last volume processed and the current time. Multiple radars can be processed simultaneously

Parameters

cfgfile [str] path of the main config file

starttime, endtime [datetime object] start and end time of the data to be processed

trajfile [str] path to file describing the trajectory

trajtype [str] type of trajectory file. Can be either 'plane' or 'lightning'

flashnr [int] If larger than 0 will select a flash in a lightning trajectory file. If 0 the data corresponding to the trajectory of all flashes will be plotted

infostr [str] Information string about the actual data processing (e.g. 'RUN57'). This string is added to product files.

MULTIPROCESSING_DSET [Bool] If true the generation of datasets at the same processing level will be parallelized

MULTIPROCESSING_PROD [Bool] If true the generation of products from each dataset will be parallelized

```
pyrad.flow.flow_control.main_rt(cfgfile_list, starttime=None, endtime=None, infostr_list=None, proc_period=60, proc_finish=None)
```

main flow control. Processes radar data in real time. The start and end processing times can be determined by the user. This function is intended for a single radar

Parameters

cfgfile_list [list of str] path of the main config files

starttime, endtime [datetime object] start and end time of the data to be processed

infostr_list [list of str] Information string about the actual data processing (e.g. 'RUN57'). This string is added to product files.

proc_period [int] period of time before starting a new processing round (seconds)

cronjob_controlled [Boolean] If True means that the program is started periodically from a cronjob and therefore finishes execution after processing

proc_finish [int or None] if set to a value the program will be forced to shut down after the value (in seconds) from start time has been exceeded

Returns

end_proc [Boolean] If true the program has ended successfully

PYRAD.PROC.PROCESS_AUX

Auxiliary functions. Functions to determine the process type, pass raw data to the product generation functions, save radar data and extract data at determined points or regions of interest.

<code>get_process_func(dataset_type, dsname)</code>	Maps the dataset type into its processing function and data set format associated.
<code>process_raw(procstatus, dscfg[, radar_list])</code>	Dummy function that returns the initial input data set
<code>process_save_radar(procstatus, dscfg[, ...])</code>	Dummy function that allows to save the entire radar object
<code>process_roi(procstatus, dscfg[, radar_list])</code>	Obtains the radar data at a region of interest.
<code>process_grid(procstatus, dscfg[, radar_list])</code>	Puts the radar data in a regular grid
<code>process_azimuthal_average(procstatus, dscfg)</code>	Averages radar data in azimuth obtaining and RHI as a result

`pyrad.proc.process_aux.get_process_func(dataset_type, dsname)`
Maps the dataset type into its processing function and data set format associated.

Parameters

dataset_type [str] data set type, i.e. 'RAW', 'SAN', etc.
dsname [str] Name of dataset

Returns

func_name [str or processing function] pyrad function used to process the data set type
dsformat [str] data set format, i.e.: 'VOL', etc.

`pyrad.proc.process_aux.process_azimuthal_average(procstatus, dscfg, radar_list=None)`
Averages radar data in azimuth obtaining and RHI as a result

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
datatype [string. Dataset keyword] The data type where we want to extract the point measurement
gridconfig [dictionary. Dataset keyword] Dictionary containing some or all of this keywords: xmin, xmax, ymin, ymax, zmin, zmax : floats
minimum and maximum horizontal distance from grid origin [km] and minimum and maximum vertical distance from grid origin [m] Defaults -40, 40, -40, 40, 0., 10000.

hres, vres [floats] horizontal and vertical grid resolution [m] Defaults 1000., 500.

latorig, lonorig, altorig [floats] latitude and longitude of grid origin [deg] and altitude of grid origin [m MSL] Defaults the latitude, longitude and altitude of the radar

wfunc [str] the weighting function used to combine the radar gates close to a grid point. Possible values BARNES, CRESSMAN, NEAREST_NEIGHBOUR Default NEAREST_NEIGHBOUR

roif_func [str] the function used to compute the region of interest. Possible values: dist_beam, constant

roi [float] the (minimum) radius of the region of interest in m. Default half the largest resolution

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the gridded data

ind_rad [int] radar index

`pyrad.proc.process_aux.process_grid(procstatus, dscfg, radar_list=None)`

Puts the radar data in a regular grid

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

gridconfig [dictionary. Dataset keyword] Dictionary containing some or all of this keywords: xmin, xmax, ymin, ymax, zmin, zmax : floats

minimum and maximum horizontal distance from grid origin [km] and minimum and maximum vertical distance from grid origin [m] Defaults -40, 40, -40, 40, 0., 10000.

hres, vres [floats] horizontal and vertical grid resolution [m] Defaults 1000., 500.

latorig, lonorig, altorig [floats] latitude and longitude of grid origin [deg] and altitude of grid origin [m MSL] Defaults the latitude, longitude and altitude of the radar

wfunc [str] the weighting function used to combine the radar gates close to a grid point. Possible values BARNES, CRESSMAN, NEAREST_NEIGHBOUR Default NEAREST_NEIGHBOUR

roif_func [str] the function used to compute the region of interest. Possible values: dist_beam, constant

roi [float] the (minimum) radius of the region of interest in m. Default half the largest resolution

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the gridded data

ind_rad [int] radar index

`pyrad.proc.process_aux.process_raw(procstatus, dscfg, radar_list=None)`

Dummy function that returns the initial input data set

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_aux.process_roi(procstatus, dscfg, radar_list=None)`

Obtains the radar data at a region of interest.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the data and metadata at the point of interest

ind_rad [int] radar index

`pyrad.proc.process_aux.process_save_radar(procstatus, dscfg, radar_list=None)`

Dummy function that allows to save the entire radar object

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

PYRAD.PROC.PROCESS_CALIB

Functions for monitoring data quality and correct bias and noise effects

<code>process_correct_bias(procstatus, dscfg[, ...])</code>	Corrects a bias on the data
<code>process_correct_noise_rho_hv(procstatus, dscfg)</code>	identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Precipitation
<code>process_gc_monitoring(procstatus, dscfg[, ...])</code>	computes ground clutter monitoring statistics
<code>process_occurrence(procstatus, dscfg[, ...])</code>	computes the frequency of occurrence of data.
<code>process_occurrence_period(procstatus, dscfg)</code>	computes the frequency of occurrence over a long period of time by adding together shorter periods
<code>process_sun_hits(procstatus, dscfg[, radar_list])</code>	monitoring of the radar using sun hits

`pyrad.proc.process_calib.process_correct_bias (procstatus, dscfg, radar_list=None)`

Corrects a bias on the data

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type to correct for bias

bias [float. Dataset keyword] The bias to be corrected [dB]. Default 0

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_calib.process_correct_noise_rho_hv (procstatus, dscfg, radar_list=None)`
identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Precipitation

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The data types used in the correction

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_calib.process_gc_monitoring(procstatus, dscfg, radar_list=None)`
computes ground clutter monitoring statistics

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

excessgatespath [str. Config keyword] The path to the gates in excess of quantile location

excessgates_fname [str. Dataset keyword] The name of the gates in excess of quantile file

datatype [list of string. Dataset keyword] The input data types

step [float. Dataset keyword] The width of the histogram bin. Default is None. In that case the default step in function `get_histogram_bins` is used

regular_grid [Boolean. Dataset keyword] Whether the radar has a Boolean grid or not. Default False

val_min [Float. Dataset keyword] Minimum value to consider that the gate has signal. Default None

filter_prec [str. Dataset keyword] Give which type of volume should be filtered. None, no filtering; keep_wet, keep wet volumes; keep_dry, keep dry volumes.

rmax_prec [float. Dataset keyword] Maximum range to consider when looking for wet gates [m]

percent_prec_max [float. Dataset keyword] Maxim percentage of wet gates to consider the volume dry

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [Radar] radar object containing histogram data

ind_rad [int] radar index

`pyrad.proc.process_calib.process_occurrence(procstatus, dscfg, radar_list=None)`
computes the frequency of occurrence of data. It looks only for gates where data is present.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

regular_grid [Boolean. Dataset keyword] Whether the radar has a Boolean grid or not. Default False

rmin, rmax [float. Dataset keyword] minimum and maximum ranges where the computation takes place. If -1 the whole range is considered. Default is -1

val_min [Float. Dataset keyword] Minimum value to consider that the gate has signal. Default None

filter_prec [str. Dataset keyword] Give which type of volume should be filtered. None, no filtering; keep_wet, keep wet volumes; keep_dry, keep dry volumes.

rmax_prec [float. Dataset keyword] Maximum range to consider when looking for wet gates [m]

percent_prec_max [float. Dataset keyword] Maxim percentage of wet gates to consider the volume dry

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_calib.process_occurrence_period` (*procstatus*, *dscfg*,
radar_list=None)

computes the frequency of occurrence over a long period of time by adding together shorter periods

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

regular_grid [Boolean. Dataset keyword] Whether the radar has a Boolean grid or not. Default False

rmin, rmax [float. Dataset keyword] minimum and maximum ranges where the computation takes place. If -1 the whole range is considered. Default is -1

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_calib.process_sun_hits` (*procstatus*, *dscfg*, *radar_list=None*)
monitoring of the radar using sun hits

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] minimum range where to look for a sun hit signal [m]. Default 50000.

hmin [float. Dataset keyword] minimum altitude where to look for a sun hit signal [m MSL]. Default 10000. The actual range from which a sun hit signal will be search will be the minimum between rmin and the range from which the altitude is higher than hmin.

delev_max [float. Dataset keyword] maximum elevation distance from nominal radar elevation where to look for a sun hit signal [deg]. Default 1.5

dazim_max [float. Dataset keyword] maximum azimuth distance from nominal radar elevation where to look for a sun hit signal [deg]. Default 1.5

elmin [float. Dataset keyword] minimum radar elevation where to look for sun hits [deg]. Default 1.

nbins_min [int. Dataset keyword.] minimum number of range bins that have to contain signal to consider the ray a potential sun hit. Default 10.

attg [float. Dataset keyword] gaseous attenuation. Default None

max_std_pwr [float. Dataset keyword] maximum standard deviation of the signal power to consider the data a sun hit [dB]. Default 2.

max_std_zdr [float. Dataset keyword] maximum standard deviation of the ZDR to consider the data a sun hit [dB]. Default 2.

az_width_co [float. Dataset keyword] co-polar antenna azimuth width (convoluted with sun width) [deg]. Default None

el_width_co [float. Dataset keyword] co-polar antenna elevation width (convoluted with sun width) [deg]. Default None

az_width_cross [float. Dataset keyword] cross-polar antenna azimuth width (convoluted with sun width) [deg]. Default None

el_width_cross [float. Dataset keyword] cross-polar antenna elevation width (convoluted with sun width) [deg]. Default None

ndays [int. Dataset keyword] number of days used in sun retrieval. Default 1

coeff_band [float. Dataset keyword] multiply coefficient to transform pulse width into receiver bandwidth

radar_list [list of Radar objects] Optional. list of radar objects

Returns

sun_hits_dict [dict] dictionary containing a radar object, a sun_hits dict and a sun_retrieval dictionary

ind_rad [int] radar index

PYRAD.PROC.PROCESS_COSMO

Functions to manage COSMO data

<code>process_cosmo(procstatus, dscfg[, radar_list])</code>	Gets COSMO data and put it in radar coordinates
<code>process_hzt(procstatus, dscfg[, radar_list])</code>	Gets iso0 degree data in HZT format and put it in radar coordinates
<code>process_cosmo_lookup_table(procstatus, dscfg)</code>	Gets COSMO data and put it in radar coordinates using look up tables computed or loaded when initializing
<code>process_hzt_lookup_table(procstatus, dscfg)</code>	Gets HZT data and put it in radar coordinates using look up tables computed or loaded when initializing
<code>process_cosmo_coord(procstatus, dscfg[, ...])</code>	Gets the COSMO indices corresponding to each cosmo coordinates
<code>process_hzt_coord(procstatus, dscfg[, ...])</code>	Gets the HZT indices corresponding to each HZT coordinates

`pyrad.proc.process_cosmo.process_cosmo (procstatus, dscfg, radar_list=None)`

Gets COSMO data and put it in radar coordinates

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] arbitrary data type

keep_in_memory [int. Dataset keyword] if set keeps the COSMO data dict, the COSMO coordinates dict and the COSMO field in radar coordinates in memory

regular_grid [int. Dataset keyword] if set it is assume that the radar has a grid constant in time and there is no need to compute a new COSMO field if the COSMO data has not changed

cosmo_type [str. Dataset keyword] name of the COSMO field to process. Default TEMP

cosmo_variables [list of strings. Dataset keyword] Py-art name of the COSMO fields. Default temperature

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_cosmo.process_cosmo_coord (procstatus, dscfg, radar_list=None)`

Gets the COSMO indices corresponding to each cosmo coordinates

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [string. Dataset keyword] arbitrary data type
 cosmopath [string. General keyword] path where to store the look up table
radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output
ind_rad [int] radar index

```
pyrad.proc.process_cosmo.process_cosmo_lookup_table(procstatus, dscfg,  
                                                    radar_list=None)
```

Gets COSMO data and put it in radar coordinates using look up tables computed or loaded when initializing

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [string. Dataset keyword] arbitrary data type
 lookup_table [int. Dataset keyword] if set a pre-computed look up table for the COSMO coordinates is loaded. Otherwise the look up table is computed taking the first radar object as reference
 regular_grid [int. Dataset keyword] if set it is assume that the radar has a grid constant in time and therefore there is no need to interpolate the COSMO field in memory to the current radar grid
 cosmo_type [str. Dataset keyword] name of the COSMO field to process. Default TEMP
 cosmo_variables [list of strings. Dataset keyword] Py-art name of the COSMO fields. Default temperature
radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output
ind_rad [int] radar index

```
pyrad.proc.process_cosmo.process_hzt(procstatus, dscfg, radar_list=None)
```

Gets iso0 degree data in HZT format and put it in radar coordinates

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [string. Dataset keyword] arbitrary data type
 keep_in_memory [int. Dataset keyword] if set keeps the COSMO data dict, the COSMO coordinates dict and the COSMO field in radar coordinates in memory
 regular_grid [int. Dataset keyword] if set it is assume that the radar has a grid constant in time and there is no need to compute a new COSMO field if the COSMO data has not changed

cosmo_type [str. Dataset keyword] name of the COSMO field to process. Default TEMP

cosmo_variables [list of strings. Dataset keyword] Py-art name of the COSMO fields.
Default temperature

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_cosmo.process_hzt_coord(procstatus, dscfg, radar_list=None)`

Gets the HZT indices corresponding to each HZT coordinates

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] arbitrary data type

cosmopath [string. General keyword] path where to store the look up table

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_cosmo.process_hzt_lookup_table(procstatus, dscfg, radar_list=None)`

Gets HZT data and put it in radar coordinates using look up tables computed or loaded when initializing

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] arbitrary data type

lookup_table [int. Dataset keyword] if set a pre-computed look up table for the COSMO coordinates is loaded. Otherwise the look up table is computed taking the first radar object as reference

regular_grid [int. Dataset keyword] if set it is assume that the radar has a grid constant in time and therefore there is no need to interpolate the COSMO field in memory to the current radar grid

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

PYRAD.PROC.PROCESS_DOPPLER

Functions for processing Doppler related parameters

<code>process_dealias_fourdd(procstatus, dscfg[, ...])</code>	Dealiases the Doppler velocity field using the 4DD technique from Curtis and Houze, 2001
<code>process_dealias_region_based(procstatus, dscfg)</code>	Dealiases the Doppler velocity field using a region based algorithm
<code>process_dealias_unwrap_phase(procstatus, dscfg)</code>	Dealiases the Doppler velocity field using multi-dimensional phase unwrapping
<code>process_wind_vel(procstatus, dscfg[, radar_list])</code>	Estimates the horizontal or vertical component of the wind from the radial velocity
<code>process_windshear(procstatus, dscfg[, ...])</code>	Estimates the wind shear from the wind velocity
<code>process_vad(procstatus, dscfg[, radar_list])</code>	Estimates vertical wind profile using the VAD (velocity Azimuth Display) technique

`pyrad.proc.process_Doppler.process_dealias_fourdd` (*procstatus*, *dscfg*,
radar_list=None)
Dealiases the Doppler velocity field using the 4DD technique from Curtis and Houze, 2001

Parameters

- procstatus** [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
- dscfg** [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 - datatype** [string. Dataset keyword] The input data type
 - filt** [int. Dataset keyword] Flag controlling Bergen and Albers filter, 1 = yes, 0 = no.
 - sign** [int. Dataset keyword] Sign convention which the radial velocities in the volume created from the sounding data will will. This should match the convention used in the radar data. A value of 1 represents when positive values velocities are towards the radar, -1 represents when negative velocities are towards the radar.
- radar_list** [list of Radar objects] Optional. list of radar objects

Returns

- new_dataset** [dict] dictionary containing the output
- ind_rad** [int] radar index

`pyrad.proc.process_Doppler.process_dealias_region_based` (*procstatus*, *dscfg*,
radar_list=None)
Dealiases the Doppler velocity field using a region based algorithm

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

interval_splits [int, optional] Number of segments to split the nyquist interval into when finding regions of similar velocity. More splits creates a larger number of initial regions which takes longer to process but may result in better dealiasing. The default value of 3 seems to be a good compromise between performance and artifact free dealiasing. This value is not used if the `interval_limits` parameter is not None.

skip_between_rays, skip_along_ray [int, optional] Maximum number of filtered gates to skip over when joining regions, gaps between region larger than this will not be connected. Parameters specify the maximum number of filtered gates between and along a ray. Set these parameters to 0 to disable unfolding across filtered gates.

centered [bool, optional] True to apply centering to each sweep after the dealiasing algorithm so that the average number of unfolding is near 0. False does not apply centering which may results in individual sweeps under or over folded by the nyquist interval.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_Doppler.process_dealias_unwrap_phase` (*procstatus*, *dscfg*,
radar_list=None)

Dealiases the Doppler velocity field using multi-dimensional phase unwrapping

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

unwrap_unit [{ 'ray', 'sweep', 'volume' }, optional] Unit to unwrap independently. 'ray' will unwrap each ray individually, 'sweep' each sweep, and 'volume' will unwrap the entire volume in a single pass. 'sweep', the default, often gives superior results when the lower sweeps of the radar volume are contaminated by clutter. 'ray' does not use the `gatefilter` parameter and rays where gates are masked will result in poor dealiasing for that ray.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_Doppler.process_vad` (*procstatus*, *dscfg*, *radar_list=None*)

Estimates vertical wind profile using the VAD (velocity Azimuth Display) technique

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_Doppler.process_wind_vel` (*procstatus, dscfg, radar_list=None*)

Estimates the horizontal or vertical component of the wind from the radial velocity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

vert_proj [Boolean] If true the vertical projection is computed. Otherwise the horizontal projection is computed

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_Doppler.process_windshear` (*procstatus, dscfg, radar_list=None*)

Estimates the wind shear from the wind velocity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

az_tol [float] The tolerance in azimuth when looking for gates on top of the gate when computation is performed

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

PYRAD.PROC.PROCESS_ECHOCCLASS

Functions for echo classification and filtering

<code>process_echo_id(procstatus, dscfg[, radar_list])</code>	identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Precipitation
<code>process_birds_id(procstatus, dscfg[, radar_list])</code>	identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Birds
<code>process_clt_to_echo_id(procstatus, dscfg[, ...])</code>	Converts clutter exit code from rad4alp into pyrad echo ID
<code>process_echo_filter(procstatus, dscfg[, ...])</code>	Masks all echo types that are not of the class specified in keyword <code>echo_type</code>
<code>process_cdf(procstatus, dscfg[, radar_list])</code>	Collects the fields necessary to compute the Cumulative Distribution Function
<code>process_filter_snr(procstatus, dscfg[, ...])</code>	filters out low SNR echoes
<code>process_filter_vel_diff(procstatus, dscfg[, ...])</code>	filters out range gates that could not be used for Doppler velocity estimation
<code>process_filter_visibility(procstatus, dscfg)</code>	filters out rays gates with low visibility and corrects the reflectivity
<code>process_outlier_filter(procstatus, dscfg[, ...])</code>	filters out gates which are outliers respect to the surrounding
<code>process_hydroclass(procstatus, dscfg[, ...])</code>	Classifies precipitation echoes
<code>process_melting_layer(procstatus, dscfg[, ...])</code>	Detects the melting layer
<code>process_zdr_column(procstatus, dscfg[, ...])</code>	Detects ZDR columns

`pyrad.proc.process_echoclass.process_birds_id(procstatus, dscfg, radar_list=None)`
 identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Birds

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [list of string. Dataset keyword] The input data types
radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output
ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_cdf(procstatus, dscfg, radar_list=None)`
 Collects the fields necessary to compute the Cumulative Distribution Function

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [list of string. Dataset keyword] The input data types
radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output
ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_clt_to_echo_id(procstatus, dscfg, radar_list=None)`

Converts clutter exit code from rad4alp into pyrad echo ID

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [list of string. Dataset keyword] The input data types
radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output
ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_echo_filter(procstatus, dscfg, radar_list=None)`
Masks all echo types that are not of the class specified in keyword `echo_type`

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [list of string. Dataset keyword] The input data types
 echo_type [int] The type of echo to keep: 1 noise, 2 clutter, 3 precipitation. Default 3
radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output
ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_echo_id(procstatus, dscfg, radar_list=None)`
identifies echoes as 0: No data, 1: Noise, 2: Clutter, 3: Precipitation

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [list of string. Dataset keyword] The input data types
radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_filter_snr(procstatus, dscfg, radar_list=None)`
filters out low SNR echoes

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

SNRmin [float. Dataset keyword] The minimum SNR to keep the data.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_filter_vel_diff(procstatus, dscfg, radar_list=None)`
filters out range gates that could not be used for Doppler velocity estimation

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

SNRmin [float. Dataset keyword] The minimum SNR to keep the data.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_filter_visibility(procstatus, dscfg, radar_list=None)`
filters out rays gates with low visibility and corrects the reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

VISmin [float. Dataset keyword] The minimum visibility to keep the data.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_hydroclass` (*procstatus, dscfg, radar_list=None*)
Classifies precipitation echoes

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

HYDRO_METHOD [string. Dataset keyword] The hydrometeor classification method.
One of the following: SEMISUPERVISED

RADARCENTROIDS [string. Dataset keyword] Used with HYDRO_METHOD
SEMISUPERVISED. The name of the radar of which the derived centroids will be used.
One of the following: A Albis, L Lema, P Plaine Morte, DX50

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_melting_layer` (*procstatus, dscfg,*
radar_list=None)

Detects the melting layer

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_outlier_filter` (*procstatus, dscfg,*
radar_list=None)

filters out gates which are outliers respect to the surrounding

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

threshold [float. Dataset keyword] The distance between the value of the examined range
gate and the median of the surrounding gates to consider the gate an outlier

nb [int. Dataset keyword] The number of neighbours (to one side) to analyse. i.e. 2 would
correspond to 24 gates

nb_min [int. Dataset keyword] Minimum number of neighbouring gates to consider the
examined gate valid

percentile_min, percentile_max [float. Dataset keyword] gates below (above) these percentiles (computed over the sweep) are considered potential outliers and further examined

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_echoclass.process_zdr_column(procstatus, dscfg, radar_list=None)`

Detects ZDR columns

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

PYRAD.PROC.PROCESS_INTERCOMP

Functions used in the inter-comparison between radars

<code>process_time_avg(procstatus, dscfg[, radar_list])</code>	computes the temporal mean of a field
<code>process_weighted_time_avg(procstatus, dscfg)</code>	computes the temporal mean of a field weighted by the reflectivity
<code>process_time_avg_flag(procstatus, dscfg[, ...])</code>	computes a flag field describing the conditions of the data used while averaging
<code>process_colocated_gates(procstatus, dscfg[, ...])</code>	Find colocated gates within two radars
<code>process_intercomp(procstatus, dscfg[, ...])</code>	intercomparison between two radars
<code>process_intercomp_time_avg(procstatus, dscfg)</code>	intercomparison between the average reflectivity of two radars

`pyrad.proc.process_intercomp.process_colocated_gates` (*procstatus*, *dscfg*,
radar_list=None)

Find colocated gates within two radars

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

h_tol [float. Dataset keyword] Tolerance in altitude difference between radar gates [m]. Default 100.

latlon_tol [float. Dataset keyword] Tolerance in latitude and longitude position between radar gates [deg]. Default 0.0005

vol_d_tol [float. Dataset keyword] Tolerance in pulse volume diameter [m]. Default 100.

vismin [float. Dataset keyword] Minimum visibility [percent]. Default None.

hmin [float. Dataset keyword] Minimum altitude [m MSL]. Default None.

hmax [float. Dataset keyword] Maximum altitude [m MSL]. Default None.

rmin [float. Dataset keyword] Minimum range [m]. Default None.

rmax [float. Dataset keyword] Maximum range [m]. Default None.

elmin [float. Dataset keyword] Minimum elevation angle [deg]. Default None.

elmax [float. Dataset keyword] Maximum elevation angle [deg]. Default None.

azrad1min [float. Dataset keyword] Minimum azimuth angle [deg] for radar 1. Default None.

azrad1max [float. Dataset keyword] Maximum azimuth angle [deg] for radar 1. Default None.

azrad2min [float. Dataset keyword] Minimum azimuth angle [deg] for radar 2. Default None.

azrad2max [float. Dataset keyword] Maximum azimuth angle [deg] for radar 2. Default None.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [radar object] radar object containing the flag field

ind_rad [int] radar index

`pyrad.proc.process_intercomp.process_intercomp(procstatus, dscfg, radar_list=None)`
intercomparison between two radars

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

coloc_data_dir [string. Dataset keyword] name of the directory containing the csv file with colocated data

coloc_radars_name [string. Dataset keyword] string identifying the radar names

azi_tol [float. Dataset keyword] azimuth tolerance between the two radars. Default 0.5 deg

ele_tol [float. Dataset keyword] elevation tolerance between the two radars. Default 0.5 deg

rng_tol [float. Dataset keyword] range tolerance between the two radars. Default 50 m

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing a dictionary with intercomparison data and the key “final” which contains a boolean that is true when all volumes have been processed

ind_rad [int] radar index

`pyrad.proc.process_intercomp.process_intercomp_time_avg(procstatus, dscfg, radar_list=None)`
intercomparison between the average reflectivity of two radars

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

coloc_data_dir [string. Dataset keyword] name of the directory containing the csv file with colocated data

coloc_radars_name [string. Dataset keyword] string identifying the radar names

azi_tol [float. Dataset keyword] azimuth tolerance between the two radars. Default 0.5 deg

ele_tol [float. Dataset keyword] elevation tolerance between the two radars. Default 0.5 deg

rng_tol [float. Dataset keyword] range tolerance between the two radars. Default 50 m

clt_max [int. Dataset keyword] maximum number of samples that can be clutter contaminated. Default 100 i.e. all

phi_excess_max [int. Dataset keyword] maximum number of samples that can have excess instantaneous PhiDP. Default 100 i.e. all

non_rain_max [int. Dataset keyword] maximum number of samples that can be no rain. Default 100 i.e. all

phi_avg_max [float. Dataset keyword] maximum average PhiDP allowed. Default 600 deg i.e. any

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing a dictionary with intercomparison data and the key “final” which contains a boolean that is true when all volumes have been processed

ind_rad [int] radar index

`pyrad.proc.process_intercomp.process_time_avg(procstatus, dscfg, radar_list=None)`
computes the temporal mean of a field

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC]. Default 0.

lin_trans: int. Dataset keyword If 1 apply linear transformation before averaging

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_intercomp.process_time_avg_flag(procstatus, dscfg, radar_list=None)`
computes a flag field describing the conditions of the data used while averaging

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC].
Default 0.

phidpmax: float. Dataset keyword maximum PhiDP

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [Radar] radar object

ind_rad [int] radar index

`pyrad.proc.process_intercomp.process_weighted_time_avg` (*procstatus*, *dscfg*,
radar_list=None)

computes the temporal mean of a field weighted by the reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

period [float. Dataset keyword] the period to average [s]. Default 3600.

start_average [float. Dataset keyword] when to start the average [s from midnight UTC].
Default 0.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [Radar] radar object

ind_rad [int] radar index

PYRAD.PROC.PROCESS_MONITORING

Functions for monitoring of the polarimetric variables

<code>process_selfconsistency_kdp_phidp(...[, ...])</code>	Computes specific differential phase and differential phase in rain using the selfconsistency between Zdr, Zh and KDP
<code>process_selfconsistency_bias(procstatus, dscfg)</code>	Estimates the reflectivity bias by means of the selfconsistency algorithm by Gourley
<code>process_estimate_phidp0(procstatus, dscfg[, ...])</code>	estimates the system differential phase offset at each ray
<code>process_rhoHV_rain(procstatus, dscfg[, ...])</code>	Keeps only suitable data to evaluate the 80 percentile of RhoHV in rain
<code>process_zdr_precip(procstatus, dscfg[, ...])</code>	Keeps only suitable data to evaluate the differential reflectivity in moderate rain or precipitation (for vertical scans)
<code>process_zdr_snow(procstatus, dscfg[, radar_list])</code>	Keeps only suitable data to evaluate the differential reflectivity in snow
<code>process_monitoring(procstatus, dscfg[, ...])</code>	computes monitoring statistics

`pyrad.proc.process_monitoring.process_estimate_phidp0` (*procstatus*, *dscfg*,
radar_list=None)
estimates the system differential phase offset at each ray

Parameters

- procstatus** [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
- dscfg** [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 - datatype** [list of string. Dataset keyword] The input data types
 - rmin** [float. Dataset keyword] The minimum range where to look for valid data [m]
 - rmax** [float. Dataset keyword] The maximum range where to look for valid data [m]
 - rcell** [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]
 - Zmin** [float. Dataset keyword] The minimum reflectivity [dBZ]
 - Zmax** [float. Dataset keyword] The maximum reflectivity [dBZ]
- radar_list** [list of Radar objects] Optional. list of radar objects

Returns

- new_dataset** [dict] dictionary containing the output
- ind_rad** [int] radar index

`pyrad.proc.process_monitoring.process_monitoring(procstatus, dscfg, radar_list=None)`
computes monitoring statistics

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [list of string. Dataset keyword] The input data types
 step [float. Dataset keyword] The width of the histogram bin. Default is None. In that case the default step in function `get_histogram_bins` is used
radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [Radar] radar object containing histogram data
ind_rad [int] radar index

`pyrad.proc.process_monitoring.process_rhohv_rain(procstatus, dscfg, radar_list=None)`
Keeps only suitable data to evaluate the 80 percentile of RhoHV in rain

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [list of string. Dataset keyword] The input data types
 rmin [float. Dataset keyword] minimum range where to look for rain [m]. Default 1000.
 rmax [float. Dataset keyword] maximum range where to look for rain [m]. Default 50000.
 Zmin [float. Dataset keyword] minimum reflectivity to consider the bin as precipitation [dBZ]. Default 20.
 Zmax [float. Dataset keyword] maximum reflectivity to consider the bin as precipitation [dBZ] Default 40.
 ml_thickness [float. Dataset keyword] assumed thickness of the melting layer. Default 700.
 fzl [float. Dataset keyword] The default freezing level height. It will be used if no temperature field name is specified or the temperature field is not in the radar object. Default 2000.
radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output
ind_rad [int] radar index

`pyrad.proc.process_monitoring.process_selfconsistency_bias(procstatus, dscfg, radar_list=None)`
Estimates the reflectivity bias by means of the selfconsistency algorithm by Gourley

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 datatype [list of string. Dataset keyword] The input data types

fzl [float. Dataset keyword] Default freezing level height. Default 2000.

rsmooth [float. Dataset keyword] length of the smoothing window [m]. Default 1000.

min_rhohv [float. Dataset keyword] minimum valid RhoHV. Default 0.92

max_phidp [float. Dataset keyword] maximum valid PhiDP [deg]. Default 20.

ml_thickness [float. Dataset keyword] Melting layer thickness [m]. Default 700.

rcell [float. Dataset keyword] length of continuous precipitation to consider the precipitation cell a valid phidp segment [m]. Default 1000.

dphidp_min [float. Dataset keyword] minimum phase shift [deg]. Default 2.

dphidp_max [float. Dataset keyword] maximum phase shift [deg]. Default 16.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_monitoring.process_selfconsistency_kdp_phidp` (*procstatus*,
dscfg,
radar_list=None)

Computes specific differential phase and differential phase in rain using the selfconsistency between Zdr, Zh and KDP

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of strings. Dataset keyword] The input data types

rsmooth [float. Dataset keyword] length of the smoothing window [m]. Default 1000.

min_rhohv [float. Dataset keyword] minimum valid RhoHV. Default 0.92

max_phidp [float. Dataset keyword] maximum valid PhiDP [deg]. Default 20.

ml_thickness [float. Dataset keyword] assumed melting layer thickness [m]. Default 700.

fzl [float. Dataset keyword] The default freezing level height. It will be used if no temperature field name is specified or the temperature field is not in the radar object. Default 2000.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_monitoring.process_zdr_precip` (*procstatus*, *dscfg*, *radar_list=None*)

Keeps only suitable data to evaluate the differential reflectivity in moderate rain or precipitation (for vertical scans)

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

ml_filter [boolean. Dataset keyword] indicates if a filter on data in and above the melting layer is applied. Default True.

rmin [float. Dataset keyword] minimum range where to look for rain [m]. Default 1000.

rmax [float. Dataset keyword] maximum range where to look for rain [m]. Default 50000.

Zmin [float. Dataset keyword] minimum reflectivity to consider the bin as precipitation [dBZ]. Default 20.

Zmax [float. Dataset keyword] maximum reflectivity to consider the bin as precipitation [dBZ] Default 22.

RhoHVmin [float. Dataset keyword] minimum RhoHV to consider the bin as precipitation Default 0.97

PhiDPmax [float. Dataset keyword] maximum PhiDP to consider the bin as precipitation [deg] Default 10.

elmax [float. Dataset keyword] maximum elevation angle where to look for precipitation [deg] Default None.

ml_thickness [float. Dataset keyword] assumed thickness of the melting layer. Default 700.

fzl [float. Dataset keyword] The default freezing level height. It will be used if no temperature field name is specified or the temperature field is not in the radar object. Default 2000.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_monitoring.process_zdr_snow(procstatus, dscfg, radar_list=None)`

Keeps only suitable data to evaluate the differential reflectivity in snow

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] minimum range where to look for rain [m]. Default 1000.

rmax [float. Dataset keyword] maximum range where to look for rain [m]. Default 50000.

Zmin [float. Dataset keyword] minimum reflectivity to consider the bin as snow [dBZ]. Default 0.

Zmax [float. Dataset keyword] maximum reflectivity to consider the bin as snow [dBZ] Default 30.

SNRmin [float. Dataset keyword] minimum SNR to consider the bin as snow [dB]. Default 10.

SNRmax [float. Dataset keyword] maximum SNR to consider the bin as snow [dB] Default 50.

RhoHVmin [float. Dataset keyword] minimum RhoHV to consider the bin as snow Default 0.97

PhiDPmax [float. Dataset keyword] maximum PhiDP to consider the bin as snow [deg] Default 10.

elmax [float. Dataset keyword] maximum elevation angle where to look for snow [deg] Default None.

KDPmax [float. Dataset keyword] maximum KDP to consider the bin as snow [deg] Default None

TEMPmin [float. Dataset keyword] minimum temperature to consider the bin as snow [deg C]. Default None

TEMPmax [float. Dataset keyword] maximum temperature to consider the bin as snow [deg C] Default None

hydroclass [list of ints. Dataset keyword] list of hydrometeor classes to keep for the analysis Default [1] (dry snow)

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

PYRAD.PROC.PROCESS_PHASE

Functions for PhiDP and KDP processing and attenuation correction

<code>process_correct_phidp0(procstatus, dscfg[, ...])</code>	corrects phidp of the system phase
<code>process_smooth_phidp_single_window(...[, ...])</code>	corrects phidp of the system phase and smoothes it using one window
<code>process_smooth_phidp_double_window(...[, ...])</code>	corrects phidp of the system phase and smoothes it using one window
<code>process_kdp_leastsquare_single_window(...[, ...])</code>	Computes specific differential phase using a piecewise least square method
<code>process_kdp_leastsquare_double_window(...[, ...])</code>	Computes specific differential phase using a piecewise least square method
<code>process_phidp_kdp_Vulpiani(procstatus, dscfg)</code>	Computes specific differential phase and differential phase using the method developed by Vulpiani et al.
<code>process_phidp_kdp_Kalman(procstatus, dscfg)</code>	Computes specific differential phase and differential phase using the Kalman filter as proposed by Schneebeli et al.
<code>process_phidp_kdp_Maesaka(procstatus, dscfg)</code>	Estimates PhiDP and KDP using the method by Maesaka.
<code>process_phidp_kdp_lp(procstatus, dscfg[, ...])</code>	Estimates PhiDP and KDP using a linear programming algorithm.
<code>process_selfconsistency_kdp_phidp</code>	
<code>process_selfconsistency_bias</code>	
<code>process_attenuation(procstatus, dscfg[, ...])</code>	Computes specific attenuation and specific differential attenuation using the Z-Phi method and corrects reflectivity and differential reflectivity

`pyrad.proc.process_phase.process_attenuation(procstatus, dscfg, radar_list=None)`

Computes specific attenuation and specific differential attenuation using the Z-Phi method and corrects reflectivity and differential reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

ATT_METHOD [float. Dataset keyword] The attenuation estimation method used. One of the following: ZPhi, Philin

fz1 [float. Dataset keyword] The default freezing level height. It will be used if no temper-

ature field name is specified or the temperature field is not in the radar object. Default 2000.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phase.process_correct_phidp0` (*procstatus*, *dscfg*, *radar_list=None*)
corrects phidp of the system phase

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] The minimum range where to look for valid data [m]

rmax [float. Dataset keyword] The maximum range where to look for valid data [m]

rcell [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]

Zmin [float. Dataset keyword] The minimum reflectivity [dBZ]

Zmax [float. Dataset keyword] The maximum reflectivity [dBZ]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phase.process_kdp_leastsquare_double_window` (*procstatus*,
dscfg,
radar_list=None)

Computes specific differential phase using a piecewise least square method

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rwinds [float. Dataset keyword] The length of the short segment for the least square method [m]

rwindl [float. Dataset keyword] The length of the long segment for the least square method [m]

Zthr [float. Dataset keyword] The threshold defining which estimated data to use [dBZ]

vectorize [Bool. Dataset keyword] Whether to vectorize the KDP processing. Default false

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phase.process_kdp_leastsquare_single_window` (*procstatus*,
dscfg,
radar_list=None)

Computes specific differential phase using a piecewise least square method

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rwind [float. Dataset keyword] The length of the segment for the least square method [m]

vectorize [bool. Dataset keyword] Whether to vectorize the KDP processing. Default false

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phase.process_phidp_kdp_Kalman` (*procstatus*,
radar_list=None) *dscfg*,

Computes specific differential phase and differential phase using the Kalman filter as proposed by Schneebeli et al. The data is assumed to be clutter free and continous

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

parallel [boolean. Dataset keyword] if set use parallel computing

get_phidp [boolean. Dataset keyword] if set the PhiDP computed by integrating the resultant KDP is added to the radar field

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phase.process_phidp_kdp_Maesaka` (*procstatus*,
radar_list=None) *dscfg*,

Estimates PhiDP and KDP using the method by Maesaka. This method only retrieves data in rain (i.e. below the melting layer)

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] The minimum range where to look for valid data [m]

rmax [float. Dataset keyword] The maximum range where to look for valid data [m]

rcell [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]

Zmin [float. Dataset keyword] The minimum reflectivity [dBZ]

Zmax [float. Dataset keyword] The maximum reflectivity [dBZ]

fzl [float. Dataset keyword] The freezing level height [m]. Default 2000.

ml_thickness [float. Dataset keyword] The melting layer thickness in meters. Default 700.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phase.process_phidp_kdp_Vulpiani` (*procstatus*, *dscfg*,
radar_list=None)

Computes specific differential phase and differential phase using the method developed by Vulpiani et al. The data is assumed to be clutter free and monotonous

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rwind [float. Dataset keyword] The length of the segment [m]

n_iter [int. Dataset keyword] number of iterations

interp [boolean. Dataset keyword] if set non valid values are interpolated using neighbouring valid values

parallel [boolean. Dataset keyword] if set use parallel computing

get_phidp [boolean. Dataset keyword] if set the PhiDP computed by integrating the resultant KDP is added to the radar field

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phase.process_phidp_kdp_lp` (*procstatus*, *dscfg*, *radar_list=None*)

Estimates PhiDP and KDP using a linear programming algorithm. This method only retrieves data in rain (i.e. below the melting layer)

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

fzl [float. Dataset keyword] The freezing level height [m]. Default 2000.

ml_thickness [float. Dataset keyword] The melting layer thickness in meters. Default 700.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phase.process_smooth_phidp_double_window` (*procstatus*, *dscfg*,
radar_list=None)
corrects phidp of the system phase and smoothes it using one window

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] The minimum range where to look for valid data [m]

rmax [float. Dataset keyword] The maximum range where to look for valid data [m]

rcell [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]

rwinds [float. Dataset keyword] The length of the short smoothing window [m]

rwindl [float. Dataset keyword] The length of the long smoothing window [m]

Zmin [float. Dataset keyword] The minimum reflectivity [dBZ]

Zmax [float. Dataset keyword] The maximum reflectivity [dBZ]

Zthr [float. Dataset keyword] The threshold defining wich smoothed data to used [dBZ]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_phase.process_smooth_phidp_single_window` (*procstatus*, *dscfg*,
radar_list=None)
corrects phidp of the system phase and smoothes it using one window

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

rmin [float. Dataset keyword] The minimum range where to look for valid data [m]

rmax [float. Dataset keyword] The maximum range where to look for valid data [m]

rcell [float. Dataset keyword] The length of a continuous cell to consider it valid precip [m]

rwind [float. Dataset keyword] The length of the smoothing window [m]

Zmin [float. Dataset keyword] The minimum reflectivity [dBZ]

Zmax [float. Dataset keyword] The maximum reflectivity [dBZ]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

PYRAD.PROC.PROCESS_RETRIEVE

Functions for retrieving new moments and products

<code>process_signal_power(procstatus, dscfg, ...)</code>	Computes the signal power in dBm
<code>process_rcs_pr(procstatus, dscfg[, radar_list])</code>	Computes the radar cross-section (assuming a point target) from radar reflectivity by first computing the received power and then the RCS from it.
<code>process_rcs(procstatus, dscfg[, radar_list])</code>	Computes the radar cross-section (assuming a point target) from radar reflectivity.
<code>process_vol_refl(procstatus, dscfg[, radar_list])</code>	Computes the volumetric reflectivity in $10\log_{10}(\text{cm}^2 \text{ km}^{-3})$
<code>process_snr(procstatus, dscfg[, radar_list])</code>	Computes SNR
<code>process_l(procstatus, dscfg[, radar_list])</code>	Computes L parameter
<code>process_cdr(procstatus, dscfg[, radar_list])</code>	Computes Circular Depolarization Ratio
<code>process_rainrate(procstatus, dscfg[, radar_list])</code>	Estimates rainfall rate from polarimetric moments
<code>process_bird_density(procstatus, dscfg, ...)</code>	Computes the bird density from the volumetric reflectivity

`pyrad.proc.process_retrieve.process_bird_density (procstatus, dscfg, radar_list=None)`

Computes the bird density from the volumetric reflectivity

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

sigma_bird [float. Dataset keyword] The bird radar cross section

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_retrieve.process_cdr (procstatus, dscfg, radar_list=None)`

Computes Circular Depolarization Ratio

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_retrieve.process_l(procstatus, dscfg, radar_list=None)`

Computes L parameter

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_retrieve.process_rainrate(procstatus, dscfg, radar_list=None)`

Estimates rainfall rate from polarimetric moments

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

RR_METHOD [string. Dataset keyword] The rainfall rate estimation method. One of the following: Z, ZPoly, KDP, A, ZKDP, ZA, hydro

alpha, beta [float] factor and exponent of the R-Var power law $R = \alpha * Var^{\beta}$. Default value depending on RR_METHOD. Z (0.0376, 0.6112), KDP (None, None), A (None, None)

alphaz, betaz [float] factor and exponent of the R-Z power law $R = \alpha * Z^{\beta}$. Default value (0.0376, 0.6112)

alphazr, betazr [float] factor and exponent of the R-Z power law $R = \alpha * Z^{\beta}$ applied to rain in method hydro. Default value (0.0376, 0.6112)

alphazs, betazs [float] factor and exponent of the R-Z power law $R = \alpha * Z^{\beta}$ applied to solid precipitation in method hydro. Default value (0.1, 0.5)

alphakdp, betakdp [float] factor and exponent of the R-KDP power law $R = \alpha * KDP^{\beta}$. Default value (None, None)

alphaa, betaa [float] factor and exponent of the R-Ah power law $R = \alpha * Ah^{\beta}$. Default value (None, None)

thresh [float] In hybrid methods, Rainfall rate threshold at which the retrieval method used changes [mm/h]. Default value depending on RR_METHOD. ZKDP 10, ZA 10, hydro 10

mp_factor [float] Factor by which the Z-R relation is multiplied in the melting layer in method hydro. Default 0.6

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_retrieve.process_rcs` (*procstatus*, *dscfg*, *radar_list=None*)

Computes the radar cross-section (assuming a point target) from radar reflectivity.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

kw2 [float. Dataset keyowrd] The water constant

pulse_width [float. Dataset keyowrd] The pulse width [s]

beamwidthv [float. Global keyword] The vertical polarization antenna beamwidth [deg].
Used if input is vertical reflectivity

beamwidthh [float. Global keyword] The horizontal polarization antenna beamwidth [deg].
Used if input is horizontal reflectivity

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_retrieve.process_rcs_pr` (*procstatus*, *dscfg*, *radar_list=None*)

Computes the radar cross-section (assuming a point target) from radar reflectivity by first computing the received power and then the RCS from it.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

antenna_gain [float. Global keyword] The antenna gain [dB]

txpwr [float. Global keyword] The transmitted power of the vertical channel [dBm]. Used
if input is vertical reflectivity

mflossv [float. Global keyword] The matching filter losses of the vertical channel. Used if
input is vertical reflectivity

radconstv [float. Global keyword] The vertical channel radar constant. Used if input is
vertical reflectivity

lrxv [float. Global keyword] The receiver losses from the antenna feed to the reference
point. [dB] positive value Used if input is vertical reflectivity

ltxv [float. Global keyword] The transmitter losses from the output of the high power am-
plifier to the antenna feed. [dB] positive value Used if input is vertical reflectivity

lradomev [float. Global keyword] The 1-way dry radome losses [dB] positive value. Used if input is vertical reflectivity

txpwrh [float. Global keyword] The transmitted power of the horizontal channel [dBm]. Used if input is horizontal reflectivity

mflossh [float. Global keyword] The matching filter losses of the vertical channel. Used if input is horizontal reflectivity

radconsth [float. Global keyword] The horizontal channel radar constant. Used if input is horizontal reflectivity

lrxh [float. Global keyword] The receiver losses from the antenna feed to the reference point. [dB] positive value Used if input is horizontal reflectivity

ltxh [float. Global keyword] The transmitter losses from the output of the high power amplifier to the antenna feed. [dB] positive value Used if input is horizontal reflectivity

lradomeh [float. Global keyword] The 1-way dry radome losses [dB] positive value. Used if input is horizontal reflectivity

attg [float. Dataset keyword] The gas attenuation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_retrieve.process_signal_power(procstatus, dscfg, radar_list=None)`

Computes the signal power in dBm

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

mflossv [float. Global keyword] The matching filter losses of the vertical channel. Used if input is vertical reflectivity

radconstv [float. Global keyword] The vertical channel radar constant. Used if input is vertical reflectivity

lrxv [float. Global keyword] The receiver losses from the antenna feed to the reference point. [dB] positive value Used if input is vertical reflectivity

lradomev [float. Global keyword] The 1-way dry radome losses [dB] positive value. Used if input is vertical reflectivity

mflossh [float. Global keyword] The matching filter losses of the vertical channel. Used if input is horizontal reflectivity

radconsth [float. Global keyword] The horizontal channel radar constant. Used if input is horizontal reflectivity

lrxh [float. Global keyword] The receiver losses from the antenna feed to the reference point. [dB] positive value Used if input is horizontal reflectivity

lradomeh [float. Global keyword] The 1-way dry radome losses [dB] positive value. Used if input is horizontal reflectivity

attg [float. Dataset keyword] The gas attenuation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_retrieve.process_snr(procstatus, dscfg, radar_list=None)`

Computes SNR

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The input data type

output_type [string. Dataset keyword] The output data type. Either SNRh or SNRv

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

`pyrad.proc.process_retrieve.process_vol_refl(procstatus, dscfg, radar_list=None)`

Computes the volumetric reflectivity in $10\log_{10}(\text{cm}^2 \text{ km}^{-3})$

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

freq [float. Dataset keyword] The radar frequency

kw [float. Dataset keyword] The water constant

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the output

ind_rad [int] radar index

PYRAD.PROC.PROCESS_TIMESERIES

Functions to obtain time series of radar data

<code>process_point_measurement(procstatus, dscfg)</code>	Obtains the radar data at a point location.
<code>process_qvp(procstatus, dscfg[, radar_list])</code>	Computes quasi vertical profiles, by averaging over height levels PPI data.
<code>process_rqvp(procstatus, dscfg[, radar_list])</code>	Computes range defined quasi vertical profiles, by averaging over height levels PPI data.
<code>process_evp(procstatus, dscfg[, radar_list])</code>	Computes enhanced vertical profiles, by averaging over height levels PPI data.
<code>process_svp(procstatus, dscfg[, radar_list])</code>	Computes slanted vertical profiles, by averaging over height levels PPI data.
<code>process_time_height(procstatus, dscfg[, ...])</code>	Produces time height radar objects at a point of interest defined by latitude and longitude.

`pyrad.proc.process_timeseries.process_evp(procstatus, dscfg, radar_list=None)`

Computes enhanced vertical profiles, by averaging over height levels PPI data.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

lat, lon [float] latitude and longitude of the point of interest [deg]

latlon_tol [float] tolerance in latitude and longitude in deg. Default 0.0005

delta_rng, delta_azi [float] maximum range distance [m] and azimuth distance [degree] from the central point of the evp containing data to average. Default 5000. and 10.

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 250.

avg_type [str] The type of averaging to perform. Can be either “mean” or “median” Default “mean”

nvalid_min [int] Minimum number of valid points to consider the data valid when performing the averaging. Default 1

interp_kind [str] type of interpolation when projecting to vertical grid: ‘none’, or ‘nearest’, etc. Default ‘none’. ‘none’ will select from all data points within the regular grid height

bin the closest to the center of the bin. 'nearest' will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the EVP and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.process_timeseries.process_point_measurement` (*procstatus*, *dscfg*,
radar_list=None)

Obtains the radar data at a point location.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

latlon [boolean. Dataset keyword] if True position is obtained from latitude, longitude information, otherwise position is obtained from antenna coordinates (range, azimuth, elevation).

truealt [boolean. Dataset keyword] if True the user input altitude is used to determine the point of interest. if False use the altitude at a given radar elevation ele over the point of interest.

lon [float. Dataset keyword] the longitude [deg]. Use when latlon is True.

lat [float. Dataset keyword] the latitude [deg]. Use when latlon is True.

alt [float. Dataset keyword] altitude [m MSL]. Use when latlon is True.

ele [float. Dataset keyword] radar elevation [deg]. Use when latlon is False or when latlon is True and truealt is False

azi [float. Dataset keyword] radar azimuth [deg]. Use when latlon is False

rng [float. Dataset keyword] range from radar [m]. Use when latlon is False

AziTol [float. Dataset keyword] azimuthal tolerance to determine which radar azimuth to use [deg]

EleTol [float. Dataset keyword] elevation tolerance to determine which radar elevation to use [deg]

RngTol [float. Dataset keyword] range tolerance to determine which radar bin to use [m]

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the data and metadata at the point of interest

ind_rad [int] radar index

`pyrad.proc.process_timeseries.process_qvp` (*procstatus*, *dscfg*, *radar_list=None*)

Computes quasi vertical profiles, by averaging over height levels PPI data.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

angle [int or float] If the radar object contains a PPI volume, the sweep number to use, if it contains an RHI volume the elevation angle. Default 0.

ang_tol [float] If the radar object contains an RHI volume, the tolerance in the elevation angle for the conversion into PPI

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 50

avg_type [str] The type of averaging to perform. Can be either “mean” or “median” Default “mean”

nvalid_min [int] Minimum number of valid points to accept average. Default 30.

interp_kind [str] type of interpolation when projecting to vertical grid: ‘none’, or ‘nearest’, etc. Default ‘none’ ‘none’ will select from all data points within the regular grid height bin the closest to the center of the bin. ‘nearest’ will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the QVP and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.process_timeseries.process_rqvp(procstatus, dscfg, radar_list=None)`

Computes range defined quasi vertical profiles, by averaging over height levels PPI data.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

anglenr [int] The sweep number to use. It assumes the radar volume consists on PPI scans

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 2.

avg_type [str] The type of averaging to perform. Can be either “mean” or “median” Default “mean”

nvalid_min [int] Minimum number of valid points to accept average. Default 30.

interp_kind [str] type of interpolation when projecting to vertical grid: ‘none’, or ‘nearest’, etc. Default ‘nearest’ ‘none’ will select from all data points within the regular grid height bin the closest to the center of the bin. ‘nearest’ will select the closest data point to

the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

rmax [float] ground range up to which the data is intended for use [m]. Default 50000.

weight_power [float] Power p of the weighting function $1/abs(grng-(rmax-1))^{**p}$ given to the data outside the desired range. -1 will set the weight to 0. Default 2.

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the QVP and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.process_timeseries.process_svp(procstatus, dscfg, radar_list=None)`

Computes slanted vertical profiles, by averaging over height levels PPI data.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

angle [int or float] If the radar object contains a PPI volume, the sweep number to use, if it contains an RHI volume the elevation angle. Default 0.

ang_tol [float] If the radar object contains an RHI volume, the tolerance in the elevation angle for the conversion into PPI

lat, lon [float] latitude and longitude of the point of interest [deg]

latlon_tol [float] tolerance in latitude and longitude in deg. Default 0.0005

delta_rng, delta_az [float] maximum range distance [m] and azimuth distance [degree] from the central point of the svp containing data to average. Default 5000. and 10.

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 250.

avg_type [str] The type of averaging to perform. Can be either “mean” or “median” Default “mean”

nvalid_min [int] Minimum number of valid points to consider the data valid when performing the averaging. Default 1

interp_kind [str] type of interpolation when projecting to vertical grid: ‘none’, or ‘nearest’, etc. Default ‘none’ ‘none’ will select from all data points within the regular grid height bin the closest to the center of the bin. ‘nearest’ will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the svp and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

`pyrad.proc.process_timeseries.process_time_height` (*procstatus*, *dscfg*,
radar_list=None)

Produces time height radar objects at a point of interest defined by latitude and longitude. A time-height contains the evolution of the vertical structure of radar measurements above the location of interest.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [string. Dataset keyword] The data type where we want to extract the point measurement

lat, lon [float] latitude and longitude of the point of interest [deg]

latlon_tol [float] tolerance in latitude and longitude in deg. Default 0.0005

hmax [float] The maximum height to plot [m]. Default 10000.

hres [float] The height resolution [m]. Default 50

interp_kind [str] type of interpolation when projecting to vertical grid: 'none', or 'nearest', etc. Default 'none' 'none' will select from all data points within the regular grid height bin the closest to the center of the bin. 'nearest' will select the closest data point to the center of the height bin regardless if it is within the height bin or not. Data points can be masked values If another type of interpolation is selected masked values will be eliminated from the data points before the interpolation

radar_list [list of Radar objects] Optional. list of radar objects

Returns

new_dataset [dict] dictionary containing the QVP and a keyboard stating whether the processing has finished or not.

ind_rad [int] radar index

PYRAD.PROC.PROCESS_TRAJ

Trajectory functions. Functions to pass trajectory dataset data to the product generation functions.

<code>process_trajectory(procstatus, dscfg[, ...])</code>	Return trajectory
<code>process_traj_trt(procstatus, dscfg[, ...])</code>	Processes data according to TRT trajectory
<code>process_traj_lightning(procstatus, dscfg[, ...])</code>	Return time series according to lightning trajectory
<code>process_traj_atplane(procstatus, dscfg[, ...])</code>	Return time series according to trajectory
<code>process_traj_antenna_pattern(procstatus, dscfg)</code>	Process a new array of data volumes considering a plane trajectory.
<code>_get_ts_values_antenna_pattern(radar, ...)</code>	Get the time series values of a trajectory using a synthetic antenna pattern
<code>_get_gates(radar, az, el, rr, tt, trajdict)</code>	Find the gates of the radar object that have to be used to compute the data of a trajectory
<code>_get_gates_trt(radar, trajectory, voltime[, ...])</code>	Find the gates of the radar object that belong to a TRT cell
<code>_get_gates_antenna_pattern(radar_sel, ...[, ...])</code>	Find the gates of the radar object that have to be used to compute the data of a trajectory as seen by another radar system
<code>_get_closests_bin(az, el, rr, tt, radar, tdict)</code>	Get the radar bin closest to a certain trajectory position
<code>_sample_out_of_sector(az, el, rr, radar_sel, ...)</code>	Check if trajectory sample is within radar sector
<code>TargetRadar(latitude, longitude, altitude)</code>	A class for dummy target radar object

class `pyrad.proc.process_traj.TargetRadar` (*latitude, longitude, altitude*)

Bases: `object`

A class for dummy target radar object

Attributes

latitude, longitude, altitude [float] Position of the dummy radar

__class__

alias of `builtins.type`

__delattr__ (*\$self, name, /*)

Implement `delattr(self, name)`.

__dict__ = `mappingproxy({'__module__': 'pyrad.proc.process_traj', '__doc__': '\n A c`

__dir__ () → list

default `dir()` implementation

__eq__ (*\$self, value, /*)

Return `self==value`.

__format__ ()
default object formatter

__ge__ (\$self, value, /)
Return self>=value.

__getattr__ (\$self, name, /)
Return getattr(self, name).

__gt__ (\$self, value, /)
Return self>value.

__hash__ (\$self, /)
Return hash(self).

__init__ (latitude, longitude, altitude)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__ ()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__ (\$self, value, /)
Return self<=value.

__lt__ (\$self, value, /)
Return self<value.

__module__ = 'pyrad.proc.process_traj'

__ne__ (\$self, value, /)
Return self!=value.

__new__ (\$type, *args, **kwargs)
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__ (\$self, /)
Return repr(self).

__setattr__ (\$self, name, value, /)
Implement setattr(self, name, value).

__sizeof__ () → int
size of object in memory, in bytes

__str__ (\$self, /)
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

`pyrad.proc.process_traj._get_closests_bin (az, el, rr, tt, radar, tdict)`

Get the radar bin closest to a certain trajectory position

Parameters

az, el, rr [floats] The trajectory position respect to the radar
tt [float] the trajectory time respect to the beginning of the radar scan
radar [radar object] the current radar object
tdict [dict] Dictionary containing trajectory parameters

Returns

radar_sel [radar object] The selected radar (Current or one of the two previous ones)
ray_sel, rr_ind [int] The selected ray and range indices of the radar field
el_vec_rnd, az_vec_rnd [array of floats] The elevation and azimuth fields of the selected radar rounded to the first decimal

`pyrad.proc.process_traj._get_gates (radar, az, el, rr, tt, trajdict, ang_tol=1.2)`

Find the gates of the radar object that have to be used to compute the data of a trajectory

Parameters

radar [radar object] The radar containing
az, el, rr [floats] The trajectory position respect to the radar
tt [float] the trajectory time respect to the beginning of the radar scan
trajdict [dict] Dictionary containing the trajectory parameters
ang_tol [float] Factor that multiplies the angle resolution. Used when determining the neighbouring rays

Returns

radar_sel [radar object] The radar volume selected as closest to trajectory point
ray_sel, rr_ind [ints] ray and range indices of the radar gate closest to the trajectory position
cell_ind [array of ints] indices of the surrounding rays
rr_min [int] index of the minimum range of the surrounding gates
rr_max [int] index of the maximum range of the surrounding gates

`pyrad.proc.process_traj._get_gates_antenna_pattern (radar_sel, target_radar,
az, rr, tt, scan_angles,
alt_tol=1000.0, latlon_tol=0.04,
max_altitude=12000.0)`

Find the gates of the radar object that have to be used to compute the data of a trajectory as seen by another radar system

Parameters

radar_sel [radar object] The radar containing real data
target_radar [radar object] The virtual radar
az, rr [floats] The trajectory position respect to the radar
tt [float] the trajectory time respect to the beginning of the radar scan
scan_angles [array] The scan angles of the virtual radar object

alt_tol [float] The tolerance in altitude [m]

latlon_tol [float] The tolerance in latitude and longitude [deg]

max_altitude [float] The maximum altitude where to look for radar data

Returns

ray_ind, rng_ind [array of ints] the indices of the radar data to use

w_ind [array of ints] The indices of the one-dimensional antenna pattern to use

`pyrad.proc.process_traj._get_gates_trt(radar, trajectory, voltime, time_tol=100.0,
alt_min=None, alt_max=None)`

Find the gates of the radar object that belong to a TRT cell

Parameters

radar [radar object] The radar containing

trajectory [trajectory object] Object containing the TRT cell position and dimensions

voltime [datetime object] The radar volume reference time

time_tol [float] Time tolerance where to look for data [s]

alt_min, alt_max [float] Minimum and maximum altitude where to look for data [m]

Returns

inds_ray, inds_rng [array of ints] The indices of the radar data inside the TRT cell

`pyrad.proc.process_traj._get_ts_values_antenna_pattern(radar, trajectory, tdict,
traj_ind, field_names)`

Get the time series values of a trajectory using a synthetic antenna pattern

Parameters

radar [radar object] The radar volume with the data

trajectory [trajectory object] The plane trajectory

tdict [dict] A dictionary containing parameters useful for trajectory computation

traj_ind [array] The indices of trajectory data within the current radar volume time

field_names [list of str] list of names of the radar field

Returns

result [Bool] A flag signaling whether radar data matching the trajectory was found

`pyrad.proc.process_traj._sample_out_of_sector(az, el, rr, radar_sel, ray_sel, rr_ind,
el_vec_rnd, az_vec_rnd)`

Check if trajectory sample is within radar sector

Parameters

az, el, rr [floats] The trajectory position respect to the radar

radar_sel [radar object] The selected radar (Current or one of the two previous ones)

ray_sel, rr_ind [int] The selected ray and range indices of the radar field

el_vec_rnd, az_vec_rnd [array of floats] The elevation and azimuth fields of the selected radar rounded to the first decimal

Returns

result [bool] False if the sample is out of sector. True otherwise

```
pyrad.proc.process_traj.process_traj_antenna_pattern (procstatus, dscfg,  
                                                    radar_list=None, trajec-  
                                                    tory=None)
```

Process a new array of data volumes considering a plane trajectory. As result a timeseries with the values transposed for a given antenna pattern is created. The result is created when the LAST flag is set.

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries]

datatype [list of string. Dataset keyword] The input data types

antennaType [str. Dataset keyword] Type of antenna of the radar we want to get the view from. Can be AZIMUTH, ELEVATION, LOWBEAM, HIGHBEAM

par_azimuth_antenna [dict. Global ekeyword] Dictionary containing the parameters of the PAR azimuth antenna, i.e. name of the file with the antenna elevation pattern and fixed antenna angle

par_elevation_antenna [dict. Global keyword] Dictionary containing the parameters of the PAR elevation antenna, i.e. name of the file with the antenna azimuth pattern and fixed antenna angle

asr_lowbeam_antenna [dict. Global keyword] Dictionary containing the parameters of the ASR low beam antenna, i.e. name of the file with the antenna elevation pattern and fixed antenna angle

asr_highbeam_antenna [dict. Global keyword] Dictionary containing the parameters of the ASR high beam antenna, i.e. name of the file with the antenna elevation pattern and fixed antenna angle

target_radar_pos [dict. Global keyword] Dictionary containing the latitude, longitude and altitude of the radar we want to get the view from. If not specifying it will assume the radar is collocated

range_all [Bool. Dataset keyword] If the real radar and the synthetic radar are co-located and this parameter is true the statistics are going to be computed using all the data from range 0 to the position of the plane. Default False

rhi_resolution [Bool. Dataset keyword] Resolution of the synthetic RHI used to compute the data as viewed from the synthetic radar [deg]. Default 0.5

max_altitude [float. Dataset keyword] Max altitude of the data to use when computing the view from the synthetic radar [m MSL]. Default 12000.

latlon_tol [float. Dataset keyword] The tolerance in latitude and longitude to determine which synthetic radar gates are co-located with real radar gates [deg]. Default 0.04

alt_tol [float. Dataset keyword] The tolerance in altitude to determine which synthetic radar gates are co-located with real radar gates [m]. Default 1000.

pattern_thres [float. Dataset keyword] The minimum of the sum of the weights given to each value in order to consider the weighted quantile valid. It is related to the number of valid data points

data_is_log [dict. Dataset keyword] Dictionary specifying for each field if it is in log (True) or linear units (False). Default False

use_nans [dict. Dataset keyword] Dictionary specifying whether the nans have to be used in the computation of the statistics for each field. Default False

nan_value [dict. Dataset keyword] Dictionary with the value to use to substitute the NaN values when computing the statistics of each field. Default 0

radar_list [list of Radar objects] Optional. list of radar objects

trajectory [Trajectory object] containing trajectory samples

Returns

trajectory [Trajectory object] Object holding time series

ind_rad [int] radar index

`pyrad.proc.process_traj.process_traj_atplane` (*procstatus, dscfg, radar_list=None, trajectory=None*)

Return time series according to trajectory

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

data_is_log [dict. Dataset keyword] Dictionary specifying for each field if it is in log (True) or linear units (False). Default False

ang_tol [float. Dataset keyword] Factor that multiplies the angle resolution. Used when determining the neighbouring rays. Default 1.2

radar_list [list of Radar objects] Optional. list of radar objects

trajectory [Trajectory object] containing trajectory samples

Returns

trajectory [Trajectory object] Object holding time series

ind_rad [int] radar index

`pyrad.proc.process_traj.process_traj_lightning` (*procstatus, dscfg, radar_list=None, trajectory=None*)

Return time series according to lightning trajectory

Parameters

procstatus [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:

datatype [list of string. Dataset keyword] The input data types

data_is_log [dict. Dataset keyword] Dictionary specifying for each field if it is in log (True) or linear units (False). Default False

ang_tol [float. Dataset keyword] Factor that multiplies the angle resolution. Used when determining the neighbouring rays. Default 1.2

radar_list [list of Radar objects] Optional. list of radar objects

trajectory [Trajectory object] containing trajectory samples

Returns

trajectory [Trajectory object] Object holding time series

ind_rad [int] radar index

`pyrad.proc.process_traj.process_traj_trt` (*procstatus*, *dscfg*, *radar_list=None*, *trajectory=None*)

Processes data according to TRT trajectory

Parameters

- procstatus** [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
- dscfg** [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 - datatype** [list of string. Dataset keyword] The input data types
 - time_tol** [float. Dataset keyword] tolerance between reference time of the radar volume and that of the TRT cell [s]. Default 100.
 - alt_min, alt_max** [float. Dataset keyword] Minimum and maximum altitude of the data inside the TRT cell to retrieve [m MSL]. Default None
- radar_list** [list of Radar objects] Optional. list of radar objects
- trajectory** [Trajectory object] containing trajectory samples

Returns

- trajectory** [Trajectory object] Object holding time series
- ind_rad** [int] radar index

`pyrad.proc.process_traj.process_trajectory` (*procstatus*, *dscfg*, *radar_list=None*, *trajectory=None*)

Return trajectory

Parameters

- procstatus** [int] Processing status: 0 initializing, 1 processing volume, 2 post-processing
- dscfg** [dictionary of dictionaries] data set configuration. Accepted Configuration Keywords:
 - datatype** [list of string. Dataset keyword] The input data types
- radar_list** [list of Radar objects] Optional. list of radar objects
- trajectory** [Trajectory object] containing trajectory samples

Returns

- new_dataset** [Trajectory object] radar object
- ind_rad** [int] None

PYRAD.PROD.PRODUCT_AUX

Auxiliary functions to generate products

<code><i>get_prodgen_func</i>(dsformat, dsname, dtype)</code>	maps the dataset format into its processing function
---	--

`pyrad.prod.product_aux.get_prodgen_func(dsformat, dsname, dtype)`
maps the dataset format into its processing function

Parameters

dsformat [str] dataset group, i.e. ‘VOL’, etc.

Returns

func [function] pyrad function used to generate the products

PYRAD.PROD.PROCESS_PRODUCT

Functions for obtaining Pyrad products from the datasets

<code>generate_occurrence_products(dataset, prdcfg)</code>	generates occurrence products
<code>generate_cosmo_coord_products(dataset, prdcfg)</code>	generates COSMO coordinates products
<code>generate_sun_hits_products(dataset, prdcfg)</code>	generates sun hits products
<code>generate_qvp_products(dataset, prdcfg)</code>	Generates quasi vertical profile products.
<code>generate_ml_products(dataset, prdcfg)</code>	Generates melting layer products.

`pyrad.prod.process_product.generate_cosmo_coord_products(dataset, prdcfg)`
generates COSMO coordinates products

Parameters

dataset [tuple] radar object containing the COSMO coordinates

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.process_product.generate_ml_products(dataset, prdcfg)`
Generates melting layer products.

Parameters

dataset [dict] dictionary containing the radar object and a keyword stating the status of the processing

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.process_product.generate_occurrence_products(dataset, prdcfg)`
generates occurrence products

Parameters

dataset [tuple] radar object and metadata dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.process_product.generate_qvp_products(dataset, prdcfg)`

Generates quasi vertical profile products. Quasi vertical profiles come from azimuthal averaging of polarimetric radar data.

Parameters

dataset [dict] dictionary containing the radar object and a keyword stating the status of the processing

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.process_product.generate_sun_hits_products(dataset, prdcfg)`

generates sun hits products

Parameters

dataset [tuple] radar object and sun hits dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

PYRAD.PROD.PROCESS_VOL_PRODUCTS

Functions for obtaining Pyrad products from a radar volume dataset

<i>generate_vol_products</i> (dataset, prdcfg)	Generates radar volume products.
--	----------------------------------

`pyrad.prod.process_vol_products.generate_vol_products` (*dataset*, *prdcfg*)
Generates radar volume products.

Parameters

dataset [dict] dictionary with key `radar_out` containing a radar object
prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

no return

PYRAD.PROD.PROCESS_GRID_PRODUCTS

Functions for obtaining Pyrad products from gridded datasets

<i>generate_grid_products</i> (dataset, prdcfg)	generates grid products
---	-------------------------

`pyrad.prod.process_grid_products.generate_grid_products` (*dataset*, *prdcfg*)
generates grid products

Parameters

dataset [grid] grid object

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

no return

`pyrad.prod.process_grid_products.generate_sparse_grid_products` (*dataset*, *prdcfg*)
generates products defined by sparse points

Parameters

dataset [dictionary containing the points and their values]

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

no return

PYRAD.PROD.PROCESS_TIMESERIES_PRODUCTS

Functions for obtaining Pyrad products from a time series datasets

<code>generate_timeseries_products(dataset, prdcfg)</code>	Generates time series products
--	--------------------------------

`pyrad.prod.process_timeseries_products.generate_timeseries_products` (*dataset*,
prdcfg)
Generates time series products

Parameters

dataset [dictionary] radar object

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

no return

PYRAD.PROD.PROCESS_MONITORING_PRODUCTS

Functions for obtaining Pyrad products from monitoring datasets

<code>generate_monitoring_products(dataset, prdcfg)</code>	generates a monitoring product
--	--------------------------------

`pyrad.prod.process_monitoring_products.generate_monitoring_products` (*dataset*,
prdcfg)
generates a monitoring product

Parameters

dataset [dictionary] dictionary containing a histogram object and some metadata
prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

PYRAD.PROD.PROCESS_INTERCOMP_PRODUCTS

Functions for obtaining Pyrad products from datasets used in the intercomparison process

<code>generate_intercomp_products(dataset, prd-</code>	Generates radar intercomparison products.
<code>cfg)</code>	

<code>generate_colocated_gates_products(dataset,</code>	Generates colocated gates products
<code>...)</code>	

<code>generate_time_avg_products(dataset, prdcfg)</code>	generates time average products
--	---------------------------------

`pyrad.prod.process_intercomp_products.generate_colocated_gates_products` (*dataset,*
prd-
cfg)

Generates colocated gates products

Parameters

dataset [tuple] radar objects and colocated gates dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.process_intercomp_products.generate_intercomp_products` (*dataset,*
prdcfg)

Generates radar intercomparison products.

Parameters

dataset [tuple] values of colocated gates dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

`pyrad.prod.process_intercomp_products.generate_time_avg_products` (*dataset, prd-*
cfg)

generates time average products

Parameters

dataset [tuple] radar objects and colocated gates dictionary

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

filename [str] the name of the file created. None otherwise

PYRAD.PROD.PROCESS_PRODUCT

Functions for obtaining Pyrad products from the datasets

<code><i>generate_traj_product</i>(traj, prdcfg)</code>	Generates trajectory products
---	-------------------------------

`pyrad.prod.process_traj_products.generate_traj_product` (*traj, prdcfg*)
Generates trajectory products

Parameters

traj [Trajectory object]

prdcfg [dictionary of dictionaries] product configuration dictionary of dictionaries

Returns

None

PYRAD.IO.IO_AUX

Auxiliary functions for reading/writing files

<code>map_hydro(hydro_data_op)</code>	maps the operational hydrometeor classification identifiers to the ones used by Py-ART
<code>map_Doppler(Doppler_data_bin, Nyquist_vel)</code>	maps the binary METRANET Doppler data to actual Doppler velocity
<code>get_save_dir(basepath, procname, dsname, prd-name)</code>	obtains the path to a product directory and eventually creates it
<code>make_filename(prdtype, dstype, dsname, ext_list)</code>	creates a product file name
<code>generate_field_name_str(datatype)</code>	Generates a field name in a nice to read format.
<code>get_datatype_metranet(datatype)</code>	maps de config file radar data type name into the corresponding metranet data type name and Py-ART field name
<code>get_datatype_odim(datatype)</code>	maps the config file radar data type name into the corresponding odim data type name and Py-ART field name
<code>get_fieldname_pyart(datatype)</code>	maps the config file radar data type name into the corresponding rainbow Py-ART field name
<code>get_fieldname_cosmo(field_name)</code>	maps the Py-ART field name into the corresponding COSMO variable name
<code>get_field_unit(datatype)</code>	Return unit of datatype.
<code>get_field_name(datatype)</code>	Return long name of datatype.
<code>get_file_list(datadescriptor, starttime, ...)</code>	gets the list of files with a time period
<code>get_trtfile_list(datapath, starttime, endtime)</code>	gets the list of TRT files with a time period
<code>get_scan_list(scandescrptor_list)</code>	determine which is the scan list for each radar
<code>get_new_rainbow_file_name(master_fname, ...)</code>	get the rainbow file name containing datatype from a master file name and data type
<code>get_datatype_fields(datadescriptor)</code>	splits the data type descriptor and provides each individual member
<code>get_dataset_fields(datasetdescr)</code>	splits the dataset type descriptor and provides each individual member
<code>get_datetime(fname, datadescriptor)</code>	Given a data descriptor gets date and time from file name
<code>find_raw_cosmo_file(voltime, datatype, cfg)</code>	Search a COSMO file in netcdf format
<code>find_cosmo_file(voltime, datatype, cfg, scanid)</code>	Search a COSMO file in Rainbow format
<code>find_hzt_file(voltime, cfg[, ind_rad])</code>	Search an ISO-0 degree file in HZT format
<code>find_rad4alpcosmo_file(voltime, datatype, ...)</code>	Search a COSMO file
<code>_get_datetime(fname, datagroup[, ftime_format])</code>	Given a data group gets date and time from file name

`pyrad.io.io_aux._get_datetime (fname, datagroup, ftime_format=None)`

Given a data group gets date and time from file name

Parameters

fname [str] file name

datadescriptor [str] radar field type. Format : [radar file type]:[datatype]

ftime_format [str or None] if the file is of type ODIM this contain the file time format

Returns

datetime [datetime object] date and time in file name

`pyrad.io.io_aux.find_cosmo_file (voltime, datatype, cfg, scanid, ind_rad=0)`

Search a COSMO file in Rainbow format

Parameters

voltime [datetime object] volume scan time

datatype [str] type of COSMO data to look for

cfg [dictionary of dictionaries] configuration info to figure out where the data is

scanid [str] name of the scan

ind_rad [int] radar index

Returns

fname [str] Name of COSMO file if it exists. None otherwise

`pyrad.io.io_aux.find_hzt_file (voltime, cfg, ind_rad=0)`

Search an ISO-0 degree file in HZT format

Parameters

voltime [datetime object] volume scan time

cfg [dictionary of dictionaries] configuration info to figure out where the data is

ind_rad [int] radar index

Returns

fname [str] Name of HZT file if it exists. None otherwise

`pyrad.io.io_aux.find_rad4alpcosmo_file (voltime, datatype, cfg, scanid, ind_rad=0)`

Search a COSMO file

Parameters

voltime [datetime object] volume scan time

datatype [str] type of COSMO data to look for

cfg: dictionary of dictionaries configuration info to figure out where the data is

ind_rad: int radar index

Returns

fname [str] Name of COSMO file if it exists. None otherwise

scanid: str name of the scan

`pyrad.io.io_aux.find_raw_cosmo_file (voltime, datatype, cfg, ind_rad=0)`

Search a COSMO file in netcdf format

Parameters

volttime [datetime object] volume scan time
datatype [str] type of COSMO data to look for
cfg [dictionary of dictionaries] configuration info to figure out where the data is
ind_rad [int] radar index

Returns

fname [str] Name of COSMO file if it exists. None otherwise

`pyrad.io.io_aux.generate_field_name_str(datatype)`

Generates a field name in a nice to read format.

Parameters

datatype [str] The data type

Returns

field_str [str] The field name

`pyrad.io.io_aux.get_dataset_fields(datasetdescr)`

splits the dataset type descriptor and provides each individual member

Parameters

datasetdescr [str] dataset type. Format : [processing level]:[dataset type]

Returns

proclevel [str] dataset processing level

dataset [str] dataset type, i.e. dBZ, ZDR, ISO0, ...

`pyrad.io.io_aux.get_datatype_fields(datadescriptor)`

splits the data type descriptor and provides each individual member

Parameters

datadescriptor [str] radar field type. Format : [radar file type]:[datatype]

Returns

radarnr [str] radar number, i.e. RADAR1, RADAR2, ...

datagroup [str] data type group, i.e. RAINBOW, RAD4ALP, ODIM, CFRADIAL, COSMO, MXPOL ...

datatype [str] data type, i.e. dBZ, ZDR, ISO0, ...

dataset [str] dataset type (for saved data only)

product [str] product type (for saved data only)

`pyrad.io.io_aux.get_datatype_metranet(datatype)`

maps de config file radar data type name into the corresponding metranet data type name and Py-ART field name

Parameters

datatype [str] config file radar data type name

Returns

metranet type [dict] dictionary containing the metranet data type name and its corresponding Py-ART field name

`pyrad.io.io_aux.get_datatype_odim(datatype)`

maps the config file radar data type name into the corresponding odim data type name and Py-ART field name

Parameters

datatype [str] config file radar data type name

Returns

metranet type [dict] dictionary containing the odim data type name and its corresponding Py-ART field name

`pyrad.io.io_aux.get_datetime(fname, datadescriptor)`

Given a data descriptor gets date and time from file name

Parameters

fname [str] file name

datadescriptor [str] radar field type. Format : [radar file type]:[datatype]

Returns

fdatetime [datetime object] date and time in file name

`pyrad.io.io_aux.get_field_name(datatype)`

Return long name of datatype.

Parameters

datatype [str] The data type

Returns

name [str] The name

`pyrad.io.io_aux.get_field_unit(datatype)`

Return unit of datatype.

Parameters

datatype [str] The data type

Returns

unit [str] The unit

`pyrad.io.io_aux.get_fieldname_cosmo(field_name)`

maps the Py-ART field name into the corresponding COSMO variable name

Parameters

field_name [str] Py-ART field name

Returns

cosmo_name [str] Py-ART variable name

`pyrad.io.io_aux.get_fieldname_pyart(datatype)`

maps the config file radar data type name into the corresponding rainbow Py-ART field name

Parameters

datatype [str] config file radar data type name

Returns

field_name [str] Py-ART field name

`pyrad.io.io_aux.get_file_list(datadescriptor, starttime, endtime, cfg, scan=None)`
gets the list of files with a time period

Parameters

datadescriptor [str] radar field type. Format : [radar file type]:[datatype]

starttime [datetime object] start of time period

endtime [datetime object] end of time period

cfg: dictionary of dictionaries configuration info to figure out where the data is

scan [str] scan name

Returns

filelist [list of strings] list of files within the time period

`pyrad.io.io_aux.get_new_rainbow_file_name(master_fname, master_datadescriptor, datatype)`
get the rainbow file name containing datatype from a master file name and data type

Parameters

master_fname [str] the master file name

master_datadescriptor [str] the master data type descriptor

datatype [str] the data type of the new file name to be created

Returns

new_fname [str] the new file name

`pyrad.io.io_aux.get_save_dir(basepath, procname, dsname, prdname, timeinfo=None, timeformat='%Y-%m-%d', create_dir=True)`
obtains the path to a product directory and eventually creates it

Parameters

basepath [str] product base path

procname [str] name of processing space

dsname [str] data set name

prdname [str] product name

timeinfo [datetime] time info to generate the date directory. If None there is no time format in the path

timeformat [str] Optional. The time format.

create_dir [boolean] If True creates the directory

Returns

savendir [str] path to product

`pyrad.io.io_aux.get_scan_list(scandescrptor_list)`
determine which is the scan list for each radar

Parameters

scandescrptor [list of string] the list of all scans for all radars

Returns

scan_list [list of lists] the list of scans corresponding to each radar

`pyrad.io.io_aux.get_trtfile_list(datapath, starttime, endtime)`
gets the list of TRT files with a time period

Parameters

datapath [str] directory where to look for data

starttime [datetime object] start of time period

endtime [datetime object] end of time period

Returns

filelist [list of strings] list of files within the time period

`pyrad.io.io_aux.make_filename(prdtype, dstype, dsname, ext_list, prdcfginfo=None, timeinfo=None, timeformat='%Y%m%d%H%M%S', runinfo=None)`
creates a product file name

Parameters

timeinfo [datetime] time info to generate the date directory

prdtype [str] product type, i.e. 'ppi', etc.

dstype [str] data set type, i.e. 'raw', etc.

dsname [str] data set name

ext_list [list of str] file name extensions, i.e. 'png'

prdcfginfo [str] Optional. string to add product configuration information, i.e. 'el0.4'

timeformat [str] Optional. The time format

runinfo [str] Optional. Additional information about the test (e.g. 'RUN01', 'TS011')

Returns

fname_list [list of str] list of file names (as many as extensions)

`pyrad.io.io_aux.map_Doppler(Doppler_data_bin, Nyquist_vel)`
maps the binary METRANET Doppler data to actual Doppler velocity

Parameters

Doppler_data_bin [numpy array] The binary METRANET data

Returns

Doppler_data [numpy array] The Doppler velocity in [m/s]

`pyrad.io.io_aux.map_hydro(hydro_data_op)`
maps the operational hydrometeor classification identifiers to the ones used by Py-ART

Parameters

hydro_data_op [numpy array] The operational hydrometeor classification data

Returns

hydro_data_py [numpy array] The pyart hydrometeor classification data

PYRAD.IO.CONFIG

Functions for reading pyrad config files

<code>read_config(fname[, cfg])</code>	Read a pyrad config file.
<code>get_num_elements(dtype, nelstr)</code>	Checks if data type is an array or a structure.
<code>string_to_datatype(dtype, strval)</code>	Converts a string containing a value into its Python value
<code>get_array(cfgfile, pos, nel, valtype)</code>	reads an array in a config file
<code>get_struct(cfgfile, pos, nels, fname)</code>	reads an struct in a config file
<code>get_array_type(dtype)</code>	Determines Python array type from the config file array type
<code>init_array(nel, dtype)</code>	Initializes a Python array

`pyrad.io.config.get_array(cfgfile, pos, nel, valtype)`
reads an array in a config file

Parameters

cfgfile [file object] config file
pos [int] position in file object
nel [int] number of elements of the ray
valtype [str] type of array

Returns

arr [array] array values
newpos [int] new position in file object

`pyrad.io.config.get_array_type(dtype)`
Determines Python array type from the config file array type

Parameters

dtype [str] config file data type

Returns

pytype [str] Python array type

`pyrad.io.config.get_num_elements(dtype, nelstr)`
Checks if data type is an array or a structure.

Parameters

dtype [str] data type specifier

nelstr [str] number of elements

Returns

nel [int] number of elements if type is *ARR or STRUCT. 0 otherwise

isstruct [bool] true if the type is STRUCT

`pyrad.io.config.get_struct(cfgfile, pos, nels, fname)`
reads an struct in a config file

Parameters

cfgfile [file object] config file

pos [int] position in file object

nel [int] number of elements of the ray

fname [str] config file name

Returns

struct [dict] dictionary of struct values

newpos [int] new position in file object

`pyrad.io.config.init_array(nel, dtype)`
Initializes a Python array

Parameters

nel [int] number of elements in the array

dtype [str] config file data type

Returns

pyarr [array] Python array

`pyrad.io.config.read_config(fname, cfg=None)`
Read a pyrad config file.

Parameters

fname [str] Name of the configuration file to read.

cfg [dict of dicts, optional] dictionary of dictionaries containing configuration parameters where the new parameters will be placed

Returns

cfg [dict of dicts] dictionary of dictionaries containing the configuration parameters

`pyrad.io.config.string_to_datatype(dtype, strval)`
Converts a string containing a value into its Python value

Parameters

dtype [str] data type specifier

strval [str] string value

Returns

val [scalar] value contained in the string

PYRAD.IO.READ_DATA_RADAR

Functions for reading radar data files

<i>get_data</i> (voltime, datatypesdescr, cfg)	Reads pyrad input data.
<i>merge_scans_rainbow</i> (basepath, scan_list, ...)	merge rainbow scans
<i>merge_scans_dem</i> (basepath, scan_list, ...)	merge rainbow scans
<i>merge_scans_rad4alp</i> (basepath, scan_list, ...)	merge rad4alp data.
<i>merge_scans_odim</i> (basepath, scan_list, ..., ...)	merge odim data.
<i>merge_scans_cosmo</i> (voltime, datatype_list, cfg)	merge rainbow scans
<i>merge_scans_cosmo_rad4alp</i> (voltime, datatype, cfg)	merge cosmo rad4alp scans.
<i>merge_scans_dem_rad4alp</i> (voltime, datatype, cfg)	merge DEM rad4alp scans.
<i>merge_scans_hydro_rad4alp</i> (voltime, datatype, cfg)	merge rad4alp hydrometeor classification scans.
<i>merge_fields_rainbow</i> (basepath, scan_name, ...)	merge Rainbow fields into a single radar object.
<i>merge_fields_cfradial</i> (basepath, loadname, ...)	merge CF/Radial fields into a single radar object.
<i>merge_fields_dem</i> (basepath, scan_name, ...)	merge DEM fields into a single radar object.
<i>merge_fields_cosmo</i> (filename_list)	merge COSMO fields in Rainbow file format
<i>get_data_rainbow</i> (filename, datatype)	gets rainbow radar data
<i>get_data_rad4alp</i> (filename, datatype_list, ...)	gets rad4alp radar data
<i>get_data_odim</i> (filename, datatype_list, ...)	gets ODIM radar data
<i>add_field</i> (radar_dest, radar_orig)	adds the fields from orig radar into dest radar.
<i>interpol_field</i> (radar_dest, radar_orig, ...)	interpolates field field_name contained in radar_orig to the grid in radar_dest

`pyrad.io.read_data_radar.add_field(radar_dest, radar_orig)`
 adds the fields from orig radar into dest radar. If they are not in the same grid, interpolates them to dest grid

Parameters

radar_dest [radar object] the destination radar

radar_orig [radar object] the radar object containing the original field

Returns

field_dest [dict] interpolated field and metadata

`pyrad.io.read_data_radar.get_data(voltime, datatypesdescr, cfg)`
 Reads pyrad input data.

Parameters

volttime [datetime object] volume scan time

datatypesdescr [list] list of radar field types to read. Format : [radar file type]:[datatype]

cfg: dictionary of dictionaries configuration info to figure out where the data is

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.get_data_mxp1(filename, datatype_list)`
gets MXP1 radar data

Parameters

filename [str] name of file containing MXP1 data

datatype_list [list of strings] list of data fields to get

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.get_data_odim(filename, datatype_list, scan_name, cfg, ind_rad=0)`
gets ODIM radar data

Parameters

filename [str] name of file containing odim data

datatype_list [list of strings] list of data fields to get

scan_name [str] name of the elevation (001 to 020)

cfg [dict] configuration dictionary

ind_rad [int] radar index

Returns

radar [Radar] radar object. None if the reading has not been successful

`pyrad.io.read_data_radar.get_data_rad4alp(filename, datatype_list, scan_name, cfg, ind_rad=0)`
gets rad4alp radar data

Parameters

filename [str] name of file containing rainbow data

datatype_list [list of strings] list of data fields to get

scan_name [str] name of the elevation (001 to 020)

cfg [dict] configuration dictionary

ind_rad [int] radar index

Returns

radar [Radar] radar object. None if the reading has not been successful

`pyrad.io.read_data_radar.get_data_rainbow(filename, datatype)`
gets rainbow radar data

Parameters

filename [str] name of file containing rainbow data

datatype [str] field name

Returns

radar [Radar or None] radar object if the reading of the data has been successful. None otherwise

`pyrad.io.read_data_radar.interpolate_field(radar_dest, radar_orig, field_name, fill_value=None, ang_tol=0.5)`
interpolates field field_name contained in radar_orig to the grid in radar_dest

Parameters

radar_dest [radar object] the destination radar

radar_orig [radar object] the radar object containing the original field

field_name: str name of the field to interpolate

fill_value: float The fill value

ang_tol [float] angle tolerance to determine whether the radar origin sweep is the radar destination sweep

Returns

field_dest [dict] interpolated field and metadata

`pyrad.io.read_data_radar.merge_fields_cfradial(basepath, loadname, voltime, datatype_list, dataset_list, product_list, rmax=0.0)`

merge CF/Radial fields into a single radar object.

Parameters

basepath [str] name of the base path where to find the data

loadname: str name of the saving directory

voltime [datetime object] reference time of the scan

datatype_list [list] list of data types to get

dataset_list [list] list of datasets that produced the data type to get. Used to get path.

product_list [list] list of products. Used to get path

rmax [float] maximum range that will be kept.

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_fields_cosmo(filename_list)`
merge COSMO fields in Rainbow file format

Parameters

filename_list [str] list of file paths where to find the data

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_fields_dem(basepath, scan_name, datatype_list)`
merge DEM fields into a single radar object.

Parameters

basepath [str] name of the base path where to find the data

scan_name: str name of the scan

datatype_list [list] lists of data types to get

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_fields_rainbow`(*basepath*, *scan_name*, *voltime*,
datatype_list)

merge Rainbow fields into a single radar object.

Parameters

basepath [str] name of the base path where to find the data

scan_name: str name of the scan

voltime [datetime object] reference time of the scan

datatype_list [list] lists of data types to get

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_scans_Doppler_rad4alp`(*voltime*, *datatype*, *cfg*,
ind_rad=0)

merge rad4alp dealised Doppler velocity scans. If data for all the scans cannot be retrieved returns None

Parameters

voltime: datetime object reference time of the scan

datatype [str] name of the data type to read

cfg [dict] configuration dictionary

ind_rad [int] radar index

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_scans_cosmo`(*voltime*, *datatype_list*, *cfg*, *ind_rad=0*)

merge rainbow scans

Parameters

voltime: datetime object reference time of the scan

datatype_list [list] lists of data types to get

cfg [dict] configuration dictionary

ind_rad [int] radar index

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_scans_cosmo_rad4alp`(*voltime*, *datatype*, *cfg*,
ind_rad=0)

merge cosmo rad4alp scans. If data for all the scans cannot be retrieved returns None

Parameters

voltime: datetime object reference time of the scan

datatype [str] name of the data type to read

cfg [dict] configuration dictionary

ind_rad [int] radar index

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_scans_dem(basepath, scan_list, datatype_list)`
merge rainbow scans

Parameters

basepath [str] base path of rad4alp radar data

scan_list [list] list of scans

datatype_list [list] lists of data types to get

radarnr [str] radar identifier number

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_scans_dem_rad4alp(voltime, datatype, cfg, ind_rad=0)`
merge DEM rad4alp scans. If data for all the scans cannot be retrieved returns None

Parameters

voltime: datetime object reference time of the scan

datatype [str] name of the data type to read

cfg [dict] configuration dictionary

ind_rad [int] radar index

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_scans_hydro_rad4alp(voltime, datatype, cfg, ind_rad=0)`
merge rad4alp hydrometeor classification scans. If data for all the scans cannot be retrieved returns None

Parameters

voltime: datetime object reference time of the scan

datatype [str] name of the data type to read

cfg [dict] configuration dictionary

ind_rad [int] radar index

Returns

radar [Radar] radar object

`pyrad.io.read_data_radar.merge_scans_mxpole(basepath, scan_list, voltime, datatype_list, cfg)`
merge rad4alp data.

Parameters

basepath [str] base path of mxpol radar data

scan_list [list] list of scans, in the case of mxpol, the elevation or azimuth denoted as 005 or 090 (for 5 or 90 degrees elevation) or 330 (for 330 degrees azimuth respectively)

volttime: datetime object reference time of the scan

datatype_list [list] lists of data types to get

cfg [dict] configuration dictionary

Returns

radar [Radar] radar object

```
pyrad.io.read_data_radar.merge_scans_odim(basepath, scan_list, radar_name, radar_res,
                                           volttime, datatype_list, dataset_list, cfg,
                                           ind_rad=0)
```

merge odim data.

Parameters

basepath [str] base path of odim radar data

scan_list [list] list of scans (h5)

volttime: datetime object reference time of the scan

datatype_list [list] lists of data types to get

dataset_list [list] list of datasets. Used to get path

cfg [dict] configuration dictionary

ind_rad [int] radar index

Returns

radar [Radar] radar object

```
pyrad.io.read_data_radar.merge_scans_rad4alp(basepath, scan_list, radar_name,
                                              radar_res, volttime, datatype_list, cfg,
                                              ind_rad=0)
```

merge rad4alp data.

Parameters

basepath [str] base path of rad4alp radar data

scan_list [list] list of scans (001 to 020)

radar_name [str] radar_name (A, D, L, ...)

radar_res [str] radar resolution (H or L)

volttime: datetime object reference time of the scan

datatype_list [list] lists of data types to get

cfg [dict] configuration dictionary

ind_rad [int] radar index

Returns

radar [Radar] radar object

```
pyrad.io.read_data_radar.merge_scans_rainbow(basepath, scan_list, volttime, scan_period,
                                              datatype_list, cfg, radarnr='RADAR001')
```

merge rainbow scans

Parameters

basepath [str] base path of rad4alp radar data

scan_list [list] list of scans

volttime: datetime object reference time of the scan

scan_period [float] time from reference time where to look for other scans data

datatype_list [list] lists of data types to get

cfg [dict] configuration dictionary

radarnr [str] radar identifier number

Returns

radar [Radar] radar object

PYRAD.IO.READ_DATA_MXPOL

Functions for reading radar mxpol data files .. autosummary:

```
:toctree: generated/
classes - MXPOL:
    pyrad_MXPOL
classes - MCH:
    pyrad_MCH
utilities - read:
    row_stack
    findTimes
    int2float_radar
    readMXPOLRadData
    readCHRadData
utilities - config:
    load_myconfig
    get_mymetadata
    get_elevation_metadata
    generate_radar_table
    generate_polvar_metadata
    convert_polvar_name
```

`pyrad.io.read_data_mxpol.convert_polvar_name` (*convention, polvar*)

Finds the correct variable name for a given convention (MXPOL, MCH) and a given variable name which was spelled with a different case or according to a different convention. For example, MXPOL convention uses 'Z' for the reflectivity variable, but if a user inserted 'Zh' this function will convert it to 'Z'. Parameters —— convention : str, destination convention; either MCH or LTE polvar : str, key of polarimetric variable to be converted Returns —— mykey : str, polarimetric variable key as used within the ProfileLab

toolbox context

`pyrad.io.read_data_mxpol.findTimes` (*num_sweep*)

Finds the times at the beginning and at the end of each sweep. Information comes from the elapsed time since the beginning of the volume scan, from the Rad4Alp: Specifications/ Request for Proposal (RFP) document.

Inputs —— num_sweep: int

rank of the sweep

elapsed_times[num_sweep][0]: float the elapsed time since the beginning of the volume scan at the beginning of the sweep

elapsed_times[num_sweep][1]: float the elapsed time since the beginning of the volume scan at the end of the sweep

`pyrad.io.read_data_mxpol.generate_polvar_metadata` (*polvar, filename=None*)

Generates a dictionary with metadata for a polarimetric variable Parameters —— polvar: str

polatimetric variable of interest

filename: str Filename of the configuration file. If None the default configuration file is loaded from the directory.

polvar_metadata: dict dictionary with metatdata for polarimetric variable of interest

`pyrad.io.read_data_mxpul.generate_radar_table(radarname, filename=None)`

Generates a table with basic radar info, based on the given (or default) configfile Parameters ——— radarname: str

name of the radar (i.e. 'ALB' or 'A', 'MXPOL' etc)

filename: str path and name of the configfile, if None, the default configfile is used

radar_table: dict table containing basic radar info

`pyrad.io.read_data_mxpul.get_elevation_metadata(radarname, filename=None)`

Gets the elevation angles for each sweep from the configuration file Inputs ——— radarname: str

name of the radar for which to retrieve elevation angles

filename: str name of the configuration file, if None, the default configuration file is used

_DEFAULT_RADAR_INFO['elevations'][radarname]: list list of elevation angles in degrees

or None if not available

`pyrad.io.read_data_mxpul.get_mymetadata(p, filename=None)`

Return a dictionary of metadata for a given parameter, p. An empty dictionary will be returned if no metadata dictionary exists for parameter p. Parameters ——— p: str

parameter name (i.e. Polvar) for which to return metadata

filename: str Filename of the configuration file. If None the default configuration file is loaded from the directory.

_DEFAULT_METADATA[p].copy(): dict a copy of the parameter of interest from the metadata dictionary

`pyrad.io.read_data_mxpul.int2float_radar(data, varname, index_angle)`

Converts radar moments from bit to float Inputs ——— data: np.array

moment data as loaded from h5 file

varname: str name of the moment (i.e. 'ZH')

index_angle: int rank of the sweep-1 (converted to base 0)

output: np.array moment data converted to float

`pyrad.io.read_data_mxpul.load_myconfig(filename=None)`

Load configuration from a config file. Parameters ——— filename: str

Filename of the configuration file. If None the default configuration file is loaded from the directory.

_DEFAULT_METADATA: dict Dictionary with metadata

```
class pyrad.io.read_data_mxppol.pyrad_IDL(filename, field_names=None, max_range=inf,  
                                           min_range=10000)  
    Bases: pyart.core.radar.Radar
```

Methods

<code>add_field(field_name, dic[, replace_existing])</code>	Add a field to the object.
<code>add_field_like(existing_field_name, ..., ...)</code>	Add a field to the object with metadata from a existing field.
<code>check_field_exists(field_name)</code>	Check that a field exists in the fields dictionary.
<code>extract_sweeps(sweeps)</code>	Create a new radar contains only the data from select sweeps.
<code>get_azimuth(sweep[, copy])</code>	Return an array of azimuth angles for a given sweep.
<code>get_elevation(sweep[, copy])</code>	Return an array of elevation angles for a given sweep.
<code>get_end(sweep)</code>	Return the ending ray for a given sweep.
<code>get_field(sweep, field_name[, copy])</code>	Return the field data for a given sweep.
<code>get_gate_x_y_z(sweep[, edges, ...])</code>	Return the x, y and z gate locations in meters for a given sweep.
<code>get_nyquist_vel(sweep[, check_uniform])</code>	Return the Nyquist velocity in meters per second for a given sweep.
<code>get_slice(sweep)</code>	Return a slice for selecting rays for a given sweep.
<code>get_start(sweep)</code>	Return the starting ray index for a given sweep.
<code>get_start_end(sweep)</code>	Return the starting and ending ray for a given sweep.
<code>info([level, out])</code>	Print information on radar.
<code>init_gate_altitude()</code>	Initialize the gate_altitude attribute.
<code>init_gate_longitude_latitude()</code>	Initialize or reset the gate_longitude and gate_latitude attributes.
<code>init_gate_x_y_z()</code>	Initialize or reset the gate_{x, y, z} attributes.
<code>init_rays_per_sweep()</code>	Initialize or reset the rays_per_sweep attribute.
<code>iter_azimuth()</code>	Return an iterator which returns sweep azimuth data.
<code>iter_elevation()</code>	Return an iterator which returns sweep elevation data.
<code>iter_end()</code>	Return an iterator over the sweep end indices.
<code>iter_field(field_name)</code>	Return an iterator which returns sweep field data.
<code>iter_slice()</code>	Return an iterator which returns sweep slice objects.
<code>iter_start()</code>	Return an iterator over the sweep start indices.
<code>iter_start_end()</code>	Return an iterator over the sweep start and end indices.

`__class__`

alias of builtins.type

`__delattr__($self, name, /)`

Implement delattr(self, name).

`__dict__ = mappingproxy({'__module__': 'pyrad.io.read_data_mxppol', '__init__': <func`

`__dir__()` → list

default dir() implementation

`__eq__($self, value, /)`

Return self==value.

`__format__()`

default object formatter

__ge__ (*\$self, value, /*)
Return self>=value.

__getattr__ (*\$self, name, /*)
Return getattr(self, name).

__getstate__ ()
Return object's state which can be pickled.

__gt__ (*\$self, value, /*)
Return self>value.

__hash__ (*\$self, /*)
Return hash(self).

__init__ (*filename, field_names=None, max_range=inf, min_range=10000*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__ ()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__ (*\$self, value, /*)
Return self<=value.

__lt__ (*\$self, value, /*)
Return self<value.

__module__ = `'pyrad.io.read_data_mxp01'`

__ne__ (*\$self, value, /*)
Return self!=value.

__new__ (*\$type, *args, **kwargs*)
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__ (*\$self, /*)
Return repr(self).

__setattr__ (*\$self, name, value, /*)
Implement setattr(self, name, value).

__setstate__ (*state*)
Restore unpicklable entries from pickled object.

__sizeof__ () → int
size of object in memory, in bytes

__str__ (*\$self, /*)
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by `abc.ABCMeta.__subclasscheck__()`. It should return `True`, `False` or `NotImplemented`. If it returns `NotImplemented`, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

`__weakref__`

list of weak references to the object (if defined)

`__check_sweep_in_range` (*sweep*)

Check that a sweep number is in range.

`__dic_info` (*attr, level, out, dic=None, ident_level=0*)

Print information on a dictionary attribute.

`add_field` (*field_name, dic, replace_existing=False*)

Add a field to the object.

Parameters

field_name [str] Name of the field to add to the dictionary of fields.

dic [dict] Dictionary contain field data and metadata.

replace_existing [bool] True to replace the existing field with key `field_name` if it exists, loosing any existing data. False will raise a `ValueError` when the field already exists.

`add_field_like` (*existing_field_name, field_name, data, replace_existing=False*)

Add a field to the object with metadata from a existing field.

Note that the data parameter is not copied by this method. If data refers to a 'data' array from an existing field dictionary, a copy should be made within or prior to using this method. If this is not done the 'data' key in both field dictionaries will point to the same NumPy array and modification of one will change the second. To copy NumPy arrays use the `copy()` method. See the Examples section for how to create a copy of the 'reflectivity' field as a field named 'reflectivity_copy'.

Parameters

existing_field_name [str] Name of an existing field to take metadata from when adding the new field to the object.

field_name [str] Name of the field to add to the dictionary of fields.

data [array] Field data. A copy of this data is not made, see the note above.

replace_existing [bool] True to replace the existing field with key `field_name` if it exists, loosing any existing data. False will raise a `ValueError` when the field already exists.

Examples

```
>>> radar.add_field_like('reflectivity', 'reflectivity_copy',
...                      radar.fields['reflectivity']['data'].copy())
```

`check_field_exists` (*field_name*)

Check that a field exists in the fields dictionary.

If the field does not exist raise a `KeyError`.

Parameters

field_name [str] Name of field to check.

`extract_sweeps` (*sweeps*)

Create a new radar contains only the data from select sweeps.

Parameters

sweeps [array_like] Sweeps (0-based) to include in new Radar object.

Returns

radar [Radar] Radar object which contains a copy of data from the selected sweeps.

get_azimuth (*sweep, copy=False*)

Return an array of azimuth angles for a given sweep.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

copy [bool, optional] True to return a copy of the azimuths. False, the default, returns a view of the azimuths (when possible), changing this data will change the data in the underlying Radar object.

Returns

azimuths [array] Array containing the azimuth angles for a given sweep.

get_elevation (*sweep, copy=False*)

Return an array of elevation angles for a given sweep.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

copy [bool, optional] True to return a copy of the elevations. False, the default, returns a view of the elevations (when possible), changing this data will change the data in the underlying Radar object.

Returns

azimuths [array] Array containing the elevation angles for a given sweep.

get_end (*sweep*)

Return the ending ray for a given sweep.

get_field (*sweep, field_name, copy=False*)

Return the field data for a given sweep.

When used with `get_gate_x_y_z()` this method can be used to obtain the data needed for plotting a radar field with the correct spatial context.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

field_name [str] Name of the field from which data should be retrieved.

copy [bool, optional] True to return a copy of the data. False, the default, returns a view of the data (when possible), changing this data will change the data in the underlying Radar object.

Returns

data [array] Array containing data for the requested sweep and field.

get_gate_x_y_z (*sweep, edges=False, filter_transitions=False*)

Return the x, y and z gate locations in meters for a given sweep.

With the default parameter this method returns the same data as contained in the `gate_x`, `gate_y` and `gate_z` attributes but this method performs the gate location calculations only for the specified sweep and therefore is more efficient than accessing this data through these attribute.

When used with `get_field()` this method can be used to obtain the data needed for plotting a radar field with the correct spatial context.

Parameters

sweep [int] Sweep number to retrieve gate locations from, 0 based.

edges [bool, optional] True to return the locations of the gate edges calculated by interpolating between the range, azimuths and elevations. False (the default) will return the locations of the gate centers with no interpolation.

filter_transitions [bool, optional] True to remove rays where the antenna was in transition between sweeps. False will include these rays. No rays will be removed if the `antenna_transition` attribute is not available (set to None).

Returns

x, y, z [2D array] Array containing the x, y and z, distances from the radar in meters for the center (or edges) for all gates in the sweep.

get_nyquist_vel (*sweep*, *check_uniform=True*)

Return the Nyquist velocity in meters per second for a given sweep.

Raises a `LookupError` if the Nyquist velocity is not available, an `Exception` is raised if the velocities are not uniform in the sweep unless `check_uniform` is set to False.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

check_uniform [bool] True to check to perform a check on the Nyquist velocities that they are uniform in the sweep, False will skip this check and return the velocity of the first ray in the sweep.

Returns

nyquist_velocity [float] Array containing the Nyquist velocity in m/s for a given sweep.

get_slice (*sweep*)

Return a slice for selecting rays for a given sweep.

get_start (*sweep*)

Return the starting ray index for a given sweep.

get_start_end (*sweep*)

Return the starting and ending ray for a given sweep.

info (*level='standard'*, *out=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Print information on radar.

Parameters

level [{ 'compact', 'standard', 'full', 'c', 's', 'f' }] Level of information on radar object to print, compact is minimal information, standard more and full everything.

out [file-like] Stream to direct output to, default is to print information to standard out (the screen).

init_gate_altitude ()

Initialize the `gate_altitude` attribute.

init_gate_longitude_latitude ()

Initialize or reset the `gate_longitude` and `gate_latitude` attributes.

init_gate_x_y_z()
Initialize or reset the gate_{x, y, z} attributes.

init_rays_per_sweep()
Initialize or reset the rays_per_sweep attribute.

iter_azimuth()
Return an iterator which returns sweep azimuth data.

iter_elevation()
Return an iterator which returns sweep elevation data.

iter_end()
Return an iterator over the sweep end indices.

iter_field(*field_name*)
Return an iterator which returns sweep field data.

iter_slice()
Return an iterator which returns sweep slice objects.

iter_start()
Return an iterator over the sweep start indices.

iter_start_end()
Return an iterator over the sweep start and end indices.

class pyrad.io.read_data_mxpole.pyrad_MCH(*filename, field_names=None, max_range=inf*)
Bases: pyart.core.radar.Radar

Methods

<code>add_field(field_name, dic[, replace_existing])</code>	Add a field to the object.
<code>add_field_like(existing_field_name, ..., [...])</code>	Add a field to the object with metadata from a existing field.
<code>check_field_exists(field_name)</code>	Check that a field exists in the fields dictionary.
<code>extract_sweeps(sweeps)</code>	Create a new radar contains only the data from select sweeps.
<code>get_azimuth(sweep[, copy])</code>	Return an array of azimuth angles for a given sweep.
<code>get_elevation(sweep[, copy])</code>	Return an array of elevation angles for a given sweep.
<code>get_end(sweep)</code>	Return the ending ray for a given sweep.
<code>get_field(sweep, field_name[, copy])</code>	Return the field data for a given sweep.
<code>get_gate_x_y_z(sweep[, edges, ...])</code>	Return the x, y and z gate locations in meters for a given sweep.
<code>get_nyquist_vel(sweep[, check_uniform])</code>	Return the Nyquist velocity in meters per second for a given sweep.
<code>get_slice(sweep)</code>	Return a slice for selecting rays for a given sweep.
<code>get_start(sweep)</code>	Return the starting ray index for a given sweep.
<code>get_start_end(sweep)</code>	Return the starting and ending ray for a given sweep.
<code>info([level, out])</code>	Print information on radar.
<code>init_gate_altitude()</code>	Initialize the gate_altitude attribute.
<code>init_gate_longitude_latitude()</code>	Initialize or reset the gate_longitude and gate_latitude attributes.
<code>init_gate_x_y_z()</code>	Initialize or reset the gate_{x, y, z} attributes.
<code>init_rays_per_sweep()</code>	Initialize or reset the rays_per_sweep attribute.

Continued on next page

Table 2 – continued from previous page

<code>iter_azimuth()</code>	Return an iterator which returns sweep azimuth data.
<code>iter_elevation()</code>	Return an iterator which returns sweep elevation data.
<code>iter_end()</code>	Return an iterator over the sweep end indices.
<code>iter_field(field_name)</code>	Return an iterator which returns sweep field data.
<code>iter_slice()</code>	Return an iterator which returns sweep slice objects.
<code>iter_start()</code>	Return an iterator over the sweep start indices.
<code>iter_start_end()</code>	Return an iterator over the sweep start and end indices.

```

__class__
    alias of builtins.type

__delattr__ ($self, name, /)
    Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'pyrad.io.read_data_mxp1', '__init__': <func
__dir__ () → list
    default dir() implementation

__eq__ ($self, value, /)
    Return self==value.

__format__ ()
    default object formatter

__ge__ ($self, value, /)
    Return self>=value.

__getattr__ ($self, name, /)
    Return getattr(self, name).

__getstate__ ()
    Return object's state which can be pickled.

__gt__ ($self, value, /)
    Return self>value.

__hash__ ($self, /)
    Return hash(self).

__init__ (filename, field_names=None, max_range=inf)
    Initialize self. See help(type(self)) for accurate signature.

__init_subclass__ ()
    This method is called when a class is subclassed.

    The default implementation does nothing. It may be overridden to extend subclasses.

__le__ ($self, value, /)
    Return self<=value.

__lt__ ($self, value, /)
    Return self<value.

__module__ = 'pyrad.io.read_data_mxp1'

__ne__ ($self, value, /)
    Return self!=value.

```

__new__ (*\$type, *args, **kwargs*)
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex ()
helper for pickle

__repr__ (*\$self, /*)
Return repr(self).

__setattr__ (*\$self, name, value, /*)
Implement setattr(self, name, value).

__setstate__ (*state*)
Restore unpicklable entries from pickled object.

__sizeof__ () → int
size of object in memory, in bytes

__str__ (*\$self, /*)
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_check_sweep_in_range (*sweep*)
Check that a sweep number is in range.

_dic_info (*attr, level, out, dic=None, ident_level=0*)
Print information on a dictionary attribute.

add_field (*field_name, dic, replace_existing=False*)
Add a field to the object.

Parameters

field_name [str] Name of the field to add to the dictionary of fields.

dic [dict] Dictionary contain field data and metadata.

replace_existing [bool] True to replace the existing field with key field_name if it exists, loosing any existing data. False will raise a ValueError when the field already exists.

add_field_like (*existing_field_name, field_name, data, replace_existing=False*)

Add a field to the object with metadata from a existing field.

Note that the data parameter is not copied by this method. If data refers to a 'data' array from an existing field dictionary, a copy should be made within or prior to using this method. If this is not done the 'data' key in both field dictionaries will point to the same NumPy array and modification of one will change the second. To copy NumPy arrays use the copy() method. See the Examples section for how to create a copy of the 'reflectivity' field as a field named 'reflectivity_copy'.

Parameters

existing_field_name [str] Name of an existing field to take metadata from when adding the new field to the object.

field_name [str] Name of the field to add to the dictionary of fields.

data [array] Field data. A copy of this data is not made, see the note above.

replace_existing [bool] True to replace the existing field with key field_name if it exists, loosing any existing data. False will raise a ValueError when the field already exists.

Examples

```
>>> radar.add_field_like('reflectivity', 'reflectivity_copy',
...                      radar.fields['reflectivity']['data'].copy())
```

check_field_exists (*field_name*)

Check that a field exists in the fields dictionary.

If the field does not exist raise a KeyError.

Parameters

field_name [str] Name of field to check.

extract_sweeps (*sweeps*)

Create a new radar contains only the data from select sweeps.

Parameters

sweeps [array_like] Sweeps (0-based) to include in new Radar object.

Returns

radar [Radar] Radar object which contains a copy of data from the selected sweeps.

get_azimuth (*sweep, copy=False*)

Return an array of azimuth angles for a given sweep.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

copy [bool, optional] True to return a copy of the azimuths. False, the default, returns a view of the azimuths (when possible), changing this data will change the data in the underlying Radar object.

Returns

azimuths [array] Array containing the azimuth angles for a given sweep.

get_elevation (*sweep, copy=False*)

Return an array of elevation angles for a given sweep.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

copy [bool, optional] True to return a copy of the elevations. False, the default, returns a view of the elevations (when possible), changing this data will change the data in the underlying Radar object.

Returns

azimuths [array] Array containing the elevation angles for a given sweep.

get_end (*sweep*)

Return the ending ray for a given sweep.

get_field (*sweep, field_name, copy=False*)

Return the field data for a given sweep.

When used with `get_gate_x_y_z()` this method can be used to obtain the data needed for plotting a radar field with the correct spatial context.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

field_name [str] Name of the field from which data should be retrieved.

copy [bool, optional] True to return a copy of the data. False, the default, returns a view of the data (when possible), changing this data will change the data in the underlying Radar object.

Returns

data [array] Array containing data for the requested sweep and field.

get_gate_x_y_z (*sweep, edges=False, filter_transitions=False*)

Return the x, y and z gate locations in meters for a given sweep.

With the default parameter this method returns the same data as contained in the `gate_x`, `gate_y` and `gate_z` attributes but this method performs the gate location calculations only for the specified sweep and therefore is more efficient than accessing this data through these attribute.

When used with `get_field()` this method can be used to obtain the data needed for plotting a radar field with the correct spatial context.

Parameters

sweep [int] Sweep number to retrieve gate locations from, 0 based.

edges [bool, optional] True to return the locations of the gate edges calculated by interpolating between the range, azimuths and elevations. False (the default) will return the locations of the gate centers with no interpolation.

filter_transitions [bool, optional] True to remove rays where the antenna was in transition between sweeps. False will include these rays. No rays will be removed if the `antenna_transition` attribute is not available (set to None).

Returns

x, y, z [2D array] Array containing the x, y and z, distances from the radar in meters for the center (or edges) for all gates in the sweep.

get_nyquist_vel (*sweep, check_uniform=True*)

Return the Nyquist velocity in meters per second for a given sweep.

Raises a `LookupError` if the Nyquist velocity is not available, an `Exception` is raised if the velocities are not uniform in the sweep unless `check_uniform` is set to False.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

check_uniform [bool] True to check to perform a check on the Nyquist velocities that they are uniform in the sweep, False will skip this check and return the velocity of the first ray in the sweep.

Returns

nyquist_velocity [float] Array containing the Nyquist velocity in m/s for a given sweep.

get_slice (*sweep*)

Return a slice for selecting rays for a given sweep.

get_start (*sweep*)

Return the starting ray index for a given sweep.

get_start_end (*sweep*)

Return the starting and ending ray for a given sweep.

info (*level='standard', out=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Print information on radar.

Parameters

level [{ 'compact', 'standard', 'full', 'c', 's', 'f' }] Level of information on radar object to print, compact is minimal information, standard more and full everything.

out [file-like] Stream to direct output to, default is to print information to standard out (the screen).

init_gate_altitude ()

Initialize the gate_altitude attribute.

init_gate_longitude_latitude ()

Initialize or reset the gate_longitude and gate_latitude attributes.

init_gate_x_y_z ()

Initialize or reset the gate_{x, y, z} attributes.

init_rays_per_sweep ()

Initialize or reset the rays_per_sweep attribute.

iter_azimuth ()

Return an iterator which returns sweep azimuth data.

iter_elevation ()

Return an iterator which returns sweep elevation data.

iter_end ()

Return an iterator over the sweep end indices.

iter_field (*field_name*)

Return an iterator which returns sweep field data.

iter_slice ()

Return an iterator which returns sweep slice objects.

iter_start ()

Return an iterator over the sweep start indices.

iter_start_end ()

Return an iterator over the sweep start and end indices.

class pyrad.io.read_data_mxpole.pyrad_MXPOL (*filename, field_names=None, max_range=inf, min_range=10000*)

Bases: pyart.core.radar.Radar

Methods

<code>add_field(field_name, dic[, replace_existing])</code>	Add a field to the object.
<code>add_field_like(existing_field_name, ..., ...)</code>	Add a field to the object with metadata from a existing field.
<code>check_field_exists(field_name)</code>	Check that a field exists in the fields dictionary.
<code>extract_sweeps(sweeps)</code>	Create a new radar contains only the data from select sweeps.
<code>get_azimuth(sweep[, copy])</code>	Return an array of azimuth angles for a given sweep.
<code>get_elevation(sweep[, copy])</code>	Return an array of elevation angles for a given sweep.
<code>get_end(sweep)</code>	Return the ending ray for a given sweep.
<code>get_field(sweep, field_name[, copy])</code>	Return the field data for a given sweep.
<code>get_gate_x_y_z(sweep[, edges, ...])</code>	Return the x, y and z gate locations in meters for a given sweep.
<code>get_nyquist_vel(sweep[, check_uniform])</code>	Return the Nyquist velocity in meters per second for a given sweep.
<code>get_slice(sweep)</code>	Return a slice for selecting rays for a given sweep.
<code>get_start(sweep)</code>	Return the starting ray index for a given sweep.
<code>get_start_end(sweep)</code>	Return the starting and ending ray for a given sweep.
<code>info([level, out])</code>	Print information on radar.
<code>init_gate_altitude()</code>	Initialize the gate_altitude attribute.
<code>init_gate_longitude_latitude()</code>	Initialize or reset the gate_longitude and gate_latitude attributes.
<code>init_gate_x_y_z()</code>	Initialize or reset the gate_{x, y, z} attributes.
<code>init_rays_per_sweep()</code>	Initialize or reset the rays_per_sweep attribute.
<code>iter_azimuth()</code>	Return an iterator which returns sweep azimuth data.
<code>iter_elevation()</code>	Return an iterator which returns sweep elevation data.
<code>iter_end()</code>	Return an iterator over the sweep end indices.
<code>iter_field(field_name)</code>	Return an iterator which returns sweep field data.
<code>iter_slice()</code>	Return an iterator which returns sweep slice objects.
<code>iter_start()</code>	Return an iterator over the sweep start indices.
<code>iter_start_end()</code>	Return an iterator over the sweep start and end indices.

`__class__`

alias of `builtins.type`

`__delattr__ ($self, name, /)`

Implement `delattr(self, name)`.

`__dict__ = mappingproxy({'__module__': 'pyrad.io.read_data_mxpole', '__init__': <func`

`__dir__ ()` → list

default `dir()` implementation

`__eq__ ($self, value, /)`

Return `self==value`.

`__format__ ()`

default object formatter

`__ge__ ($self, value, /)`

Return `self>=value`.

`__getattr__ ($self, name, /)`

Return `getattr(self, name)`.

__getstate__ ()
Return object's state which can be pickled.

__gt__ (\$self, value, /)
Return self>value.

__hash__ (\$self, /)
Return hash(self).

__init__ (filename, field_names=None, max_range=inf, min_range=10000)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__ ()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__ (\$self, value, /)
Return self<=value.

__lt__ (\$self, value, /)
Return self<value.

__module__ = 'pyrad.io.read_data_mxp01'

__ne__ (\$self, value, /)
Return self!=value.

__new__ (\$type, *args, **kwargs)
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__ (\$self, /)
Return repr(self).

__setattr__ (\$self, name, value, /)
Implement setattr(self, name, value).

__setstate__ (state)
Restore unpicklable entries from pickled object.

__sizeof__ () → int
size of object in memory, in bytes

__str__ (\$self, /)
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

__check_sweep_in_range (sweep)
Check that a sweep number is in range.

_dic_info (*attr, level, out, dic=None, ident_level=0*)

Print information on a dictionary attribute.

add_field (*field_name, dic, replace_existing=False*)

Add a field to the object.

Parameters

field_name [str] Name of the field to add to the dictionary of fields.

dic [dict] Dictionary contain field data and metadata.

replace_existing [bool] True to replace the existing field with key field_name if it exists, loosing any existing data. False will raise a ValueError when the field already exists.

add_field_like (*existing_field_name, field_name, data, replace_existing=False*)

Add a field to the object with metadata from a existing field.

Note that the data parameter is not copied by this method. If data refers to a 'data' array from an existing field dictionary, a copy should be made within or prior to using this method. If this is not done the 'data' key in both field dictionaries will point to the same NumPy array and modification of one will change the second. To copy NumPy arrays use the copy() method. See the Examples section for how to create a copy of the 'reflectivity' field as a field named 'reflectivity_copy'.

Parameters

existing_field_name [str] Name of an existing field to take metadata from when adding the new field to the object.

field_name [str] Name of the field to add to the dictionary of fields.

data [array] Field data. A copy of this data is not made, see the note above.

replace_existing [bool] True to replace the existing field with key field_name if it exists, loosing any existing data. False will raise a ValueError when the field already exists.

Examples

```
>>> radar.add_field_like('reflectivity', 'reflectivity_copy',
...                      radar.fields['reflectivity']['data'].copy())
```

check_field_exists (*field_name*)

Check that a field exists in the fields dictionary.

If the field does not exist raise a KeyError.

Parameters

field_name [str] Name of field to check.

extract_sweeps (*sweeps*)

Create a new radar contains only the data from select sweeps.

Parameters

sweeps [array_like] Sweeps (0-based) to include in new Radar object.

Returns

radar [Radar] Radar object which contains a copy of data from the selected sweeps.

get_azimuth (*sweep, copy=False*)

Return an array of azimuth angles for a given sweep.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

copy [bool, optional] True to return a copy of the azimuths. False, the default, returns a view of the azimuths (when possible), changing this data will change the data in the underlying Radar object.

Returns

azimuths [array] Array containing the azimuth angles for a given sweep.

get_elevation (*sweep*, *copy=False*)

Return an array of elevation angles for a given sweep.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

copy [bool, optional] True to return a copy of the elevations. False, the default, returns a view of the elevations (when possible), changing this data will change the data in the underlying Radar object.

Returns

azimuths [array] Array containing the elevation angles for a given sweep.

get_end (*sweep*)

Return the ending ray for a given sweep.

get_field (*sweep*, *field_name*, *copy=False*)

Return the field data for a given sweep.

When used with `get_gate_x_y_z()` this method can be used to obtain the data needed for plotting a radar field with the correct spatial context.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

field_name [str] Name of the field from which data should be retrieved.

copy [bool, optional] True to return a copy of the data. False, the default, returns a view of the data (when possible), changing this data will change the data in the underlying Radar object.

Returns

data [array] Array containing data for the requested sweep and field.

get_gate_x_y_z (*sweep*, *edges=False*, *filter_transitions=False*)

Return the x, y and z gate locations in meters for a given sweep.

With the default parameter this method returns the same data as contained in the `gate_x`, `gate_y` and `gate_z` attributes but this method performs the gate location calculations only for the specified sweep and therefore is more efficient than accessing this data through these attribute.

When used with `get_field()` this method can be used to obtain the data needed for plotting a radar field with the correct spatial context.

Parameters

sweep [int] Sweep number to retrieve gate locations from, 0 based.

edges [bool, optional] True to return the locations of the gate edges calculated by interpolating between the range, azimuths and elevations. False (the default) will return the locations of the gate centers with no interpolation.

filter_transitions [bool, optional] True to remove rays where the antenna was in transition between sweeps. False will include these rays. No rays will be removed if the `antenna_transition` attribute is not available (set to None).

Returns

x, y, z [2D array] Array containing the x, y and z, distances from the radar in meters for the center (or edges) for all gates in the sweep.

get_nyquist_vel (*sweep*, *check_uniform=True*)

Return the Nyquist velocity in meters per second for a given sweep.

Raises a `LookupError` if the Nyquist velocity is not available, an `Exception` is raised if the velocities are not uniform in the sweep unless `check_uniform` is set to False.

Parameters

sweep [int] Sweep number to retrieve data for, 0 based.

check_uniform [bool] True to check to perform a check on the Nyquist velocities that they are uniform in the sweep, False will skip this check and return the velocity of the first ray in the sweep.

Returns

nyquist_velocity [float] Array containing the Nyquist velocity in m/s for a given sweep.

get_slice (*sweep*)

Return a slice for selecting rays for a given sweep.

get_start (*sweep*)

Return the starting ray index for a given sweep.

get_start_end (*sweep*)

Return the starting and ending ray for a given sweep.

info (*level='standard'*, *out=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Print information on radar.

Parameters

level [{ 'compact', 'standard', 'full', 'c', 's', 'f' }] Level of information on radar object to print, compact is minimal information, standard more and full everything.

out [file-like] Stream to direct output to, default is to print information to standard out (the screen).

init_gate_altitude ()

Initialize the `gate_altitude` attribute.

init_gate_longitude_latitude ()

Initialize or reset the `gate_longitude` and `gate_latitude` attributes.

init_gate_x_y_z ()

Initialize or reset the `gate_{x, y, z}` attributes.

init_rays_per_sweep ()

Initialize or reset the `rays_per_sweep` attribute.

iter_azimuth ()

Return an iterator which returns sweep azimuth data.

iter_elevation()
Return an iterator which returns sweep elevation data.

iter_end()
Return an iterator over the sweep end indices.

iter_field(*field_name*)
Return an iterator which returns sweep field data.

iter_slice()
Return an iterator which returns sweep slice objects.

iter_start()
Return an iterator over the sweep start indices.

iter_start_end()
Return an iterator over the sweep start and end indices.

`pyrad.io.read_data_mxpola.readCHRadData` (*filename*, *radar_name*, *variableList*, *radial_resolution*, *max_range=inf*, *min_range=0*)
Reads a HDF5 file containing processed radar data in polar coordinates Parameters ——— filename: str
complete path of the file

radar_name: str name of MCH radar

variableList: list list of variables to be read

radial_resolution: float resolution of the radar in metres (i.e. high: 83.3, low: 500.)

max_range: float maximum range upto which to read data

min_range: float minimum range from which to read data

varPol: dict the projected variables, the azimuth and the range

`pyrad.io.read_data_mxpola.readIDLRadData` (*filename*, *variableList*, *max_range=inf*, *min_range=0*)
Reads a netcdf containing IDL processed radar data in polar coordinates Parameters ——— filename: str
complete path of the file

variableList: list list of variables to be read

varPol: dict dictionary containing the variables, the azimuth and the range

metadata: dict dictionary containing the metadata of the file

`pyrad.io.read_data_mxpola.readMXPOLRadData` (*filename*, *variableList*, *max_range=inf*, *min_range=0*)
Reads a netcdf containing processed radar data in polar coordinates Parameters ——— filename: str
complete path of the file

variableList: list list of variables to be read

varPol: dict dictionary containing the variables, the azimuth and the range

metadata: dict dictionary containing the metadata of the file

`pyrad.io.read_data_mxpole.row_stack(a1, a2)`

Stacks data from subsequent sweeps, while padding “empty” columns from subsequent sweeps. Inputs —— a1:
np.array

destination array

a2: np.array array which is added onto the first array

out: np.array stacked destination and additional array, with uniform shape

Created on Wed Dec 7 10:48:31 2016

@author: fvanden

Configuration file for mxpol pyart.core.Radar class. Some information may be redundant because this file is a copy from the ProfileLab toolkit.

Functions to retrieve data from this file may be found in `pyrad.io.read_data_mxpole` under the utilities section

PYRAD.IO.READ_DATA_COSMO

Functions for reading COSMO data

<code>cosmo2radar_data(radar, cosmo_coord, cosmo_data)</code>	get the COSMO value corresponding to each radar gate using nearest neighbour interpolation
<code>cosmo2radar_coord(radar, cosmo_coord[, ...])</code>	Given the radar coordinates find the nearest COSMO model pixel
<code>get_cosmo_fields(cosmo_data, cosmo_ind[, ...])</code>	Get the COSMO data corresponding to each radar gate using a precomputed look up table of the nearest neighbour
<code>read_cosmo_data(fname[, field_names, celsius])</code>	Reads COSMO data from a netcdf file
<code>read_cosmo_coord(fname[, zmin])</code>	Reads COSMO coordinates from a netcdf file
<code>_ncvar_to_dict(ncvar[, dtype])</code>	Convert a NetCDF Dataset variable to a dictionary.
<code>_prepare_for_interpolation(x_radar, y_radar, ...)</code>	prepares the COSMO 3D volume for interpolation:
<code>_put_radar_in_swiss_coord(radar)</code>	puts the Cartesian grid of the radar coordinates in Swiss coordinates

`pyrad.io.read_data_cosmo._ncvar_to_dict(ncvar, dtype='float64')`

Convert a NetCDF Dataset variable to a dictionary.

`pyrad.io.read_data_cosmo._prepare_for_interpolation(x_radar, y_radar, z_radar, cosmo_coord, slice_xy=True, slice_z=False)`

prepares the COSMO 3D volume for interpolation:

1. if set slices the cosmo data to the area (or volume)

covered by the radar

2. creates the x, y, z grid for the interpolation

Parameters

x_radar, y_radar, z_radar [arrays] The Swiss coordinates of the radar

cosmo_coord [dict] dictionary containing the COSMO coordinates

slice_xy [boolean] if true the horizontal plane of the COSMO field is cut to the dimensions of the radar field

slice_z [boolean] if true the vertical plane of the COSMO field is cut to the dimensions of the radar field

Returns

x_cosmo, y_cosmo, z_cosmo [1D arrays] arrays containing the flatten swiss coordinates of the COSMO data in the area of interest

ind_xmin, ind_ymin, ind_zmin, ind_xmax, ind_ymax, ind_zmax [ints] the minimum and maximum indices of each dimension

`pyrad.io.read_data_cosmo._put_radar_in_swiss_coord(radar)`

puts the Cartesian grid of the radar coordinates in Swiss coordinates

Parameters

radar [Radar] the radar object containing the information on the position of the radar gates

Returns

x_radar, y_radar, z_radar [2D arrays] arrays containing swiss coordinates of the radar [in m]

`pyrad.io.read_data_cosmo.cosmo2radar_coord(radar, cosmo_coord, slice_xy=True, slice_z=False, field_name=None)`

Given the radar coordinates find the nearest COSMO model pixel

Parameters

radar [Radar] the radar object containing the information on the position of the radar gates

cosmo_coord [dict] dictionary containing the COSMO coordinates

slice_xy [boolean] if true the horizontal plane of the COSMO field is cut to the dimensions of the radar field

slice_z [boolean] if true the vertical plane of the COSMO field is cut to the dimensions of the radar field

field_name [str] name of the field

Returns

cosmo_ind_field [dict] dictionary containing a field of COSMO indices and metadata

`pyrad.io.read_data_cosmo.cosmo2radar_data(radar, cosmo_coord, cosmo_data, time_index=0, slice_xy=True, slice_z=False, field_names=['temperature'])`

get the COSMO value corresponding to each radar gate using nearest neighbour interpolation

Parameters

radar [Radar] the radar object containing the information on the position of the radar gates

cosmo_coord [dict] dictionary containing the COSMO coordinates

cosmo_data [dict] dictionary containing the COSMO data

time_index [int] index of the forecasted data

slice_xy [boolean] if true the horizontal plane of the COSMO field is cut to the dimensions of the radar field

slice_z [boolean] if true the vertical plane of the COSMO field is cut to the dimensions of the radar field

field_names [str] names of COSMO fields to convert (default temperature)

Returns

cosmo_fields [list of dict] list of dictionary with the COSMO fields and metadata

```
pyrad.io.read_data_cosmo.get_cosmo_fields(cosmo_data, cosmo_ind, time_index=0,  
                                           field_names=['temperature'])
```

Get the COSMO data corresponding to each radar gate using a precomputed look up table of the nearest neighbour

Parameters

cosmo_data [dict] dictionary containing the COSMO data and metadata
cosmo_ind [dict] dictionary containing a field of COSMO indices and metadata
time_index [int] index of the forecasted data
field_names [str] names of COSMO parameters (default temperature)

Returns

cosmo_fields [list of dict] dictionary with the COSMO fields and metadata

```
pyrad.io.read_data_cosmo.read_cosmo_coord(fname, zmin=None)
```

Reads COSMO coordinates from a netcdf file

Parameters

fname [str] name of the file to read

Returns

cosmo_coord [dictionary] dictionary with the data and metadata

```
pyrad.io.read_data_cosmo.read_cosmo_data(fname, field_names=['temperature'],  
                                          celsius=True)
```

Reads COSMO data from a netcdf file

Parameters

fname [str] name of the file to read
field_names [str] name of the variable to read
celsius [Boolean] if True and variable temperature converts data from Kelvin to Centigrade

Returns

cosmo_data [dictionary] dictionary with the data and metadata

PYRAD.IO.READ_DATA_HZT

Functions for reading HZT data

<code>hzt2radar_data(radar, hzt_coord, hzt_data[, ...])</code>	get the HZT value corresponding to each radar gate using nearest neighbour interpolation
<code>hzt2radar_coord(radar, hzt_coord[, ...])</code>	Given the radar coordinates find the nearest HZT pixel
<code>get_iso0_field(hzt_data, hzt_ind, z_radar[, ...])</code>	Get the height over iso0 data corresponding to each radar gate using a precomputed look up table of the nearest neighbour
<code>read_hzt_data(fname[, chy0, chx0])</code>	Reads iso-0 degree data from an HZT file
<code>_prepare_for_interpolation(x_radar, y_radar, ...)</code>	prepares the HZT 2D volume for interpolation:

```
pyrad.io.read_data_hzt._prepare_for_interpolation(x_radar, y_radar, hzt_coord,
                                                    slice_xy=True)
```

prepares the HZT 2D volume for interpolation:

1. if set slices the cosmo data to the area covered by the radar
2. creates the x, y grid for the interpolation

Parameters

x_radar, y_radar [arrays] The Swiss coordinates of the radar

hzt_coord [dict] dictionary containing the HZT coordinates

slice_xy [boolean] if true the horizontal plane of the HZT field is cut to the dimensions of the radar field

Returns

x_hzt, y_hzt [1D arrays] arrays containing the flatten swiss coordinates of the HZT data in the area of interest [m]

ind_xmin, ind_ymin, ind_xmax, ind_ymax [ints] the minimum and maximum indices of each dimension

```
pyrad.io.read_data_hzt.get_iso0_field(hzt_data, hzt_ind, z_radar,
                                       field_name='height_over_iso0')
```

Get the height over iso0 data corresponding to each radar gate using a precomputed look up table of the nearest neighbour

Parameters

hzt_data [dict] dictionary containing the HZT data and metadata

hzt_ind [dict] dictionary containing a field of HZT indices and metadata

z_radar [ndarray] gates altitude [m MSL]

field_name [str] names of HZT parameters (default height_over_iso0)

Returns

iso0_field [list of dict] dictionary with the height over iso0 field and metadata

```
pyrad.io.read_data_hzt.hzt2radar_coord(radar, hzt_coord, slice_xy=True,
                                         field_name=None)
```

Given the radar coordinates find the nearest HZT pixel

Parameters

radar [Radar] the radar object containing the information on the position of the radar gates

hzt_coord [dict] dictionary containing the HZT coordinates

slice_xy [boolean] if true the horizontal plane of the HZT field is cut to the dimensions of the radar field

field_name [str] name of the field

Returns

hzt_ind_field [dict] dictionary containing a field of HZT indices and metadata

```
pyrad.io.read_data_hzt.hzt2radar_data(radar, hzt_coord, hzt_data, slice_xy=True,
                                         field_name='height_over_iso0')
```

get the HZT value corresponding to each radar gate using nearest neighbour interpolation

Parameters

radar [Radar] the radar object containing the information on the position of the radar gates

hzt_coord [dict] dictionary containing the HZT coordinates

hzt_data [dict] dictionary containing the HZT data

slice_xy [boolean] if true the horizontal plane of the COSMO field is cut to the dimensions of the radar field

field_name [str] name of HZT fields to convert (default height_over_iso0)

Returns

hzt_fields [list of dict] list of dictionary with the HZT fields and metadata

```
pyrad.io.read_data_hzt.read_hzt_data(fname, chy0=255.0, chx0=-160.0)
```

Reads iso-0 degree data from an HZT file

Parameters

fname [str] name of the file to read

chy0, chx0: south west point of grid in Swiss coordinates [km]

Returns

hzt_data [dictionary] dictionary with the data and metadata

PYRAD.IO.READ_DATA_SENSOR

Functions for reading data from other sensors

<code>read_trt_scores(fname)</code>	Reads the TRT scores contained in a text file.
<code>read_trt_cell_lightning(fname)</code>	Reads the lightning data of a TRT cell.
<code>read_trt_data(fname)</code>	Reads the TRT data contained in a text file.
<code>read_trt_traj_data(fname)</code>	Reads the TRT cell data contained in a text file.
<code>read_lightning(fname[, filter_data])</code>	Reads lightning data contained in a text file.
<code>read_meteorage(fname)</code>	Reads METEORAGE lightning data contained in a text file.
<code>read_lightning_traj(fname)</code>	Reads lightning trajectory data contained in a csv file.
<code>read_lightning_all(fname[, labels])</code>	Reads a file containing lightning data and co-located polarimetric data.
<code>get_sensor_data(date, datatype, cfg)</code>	Gets data from a point measurement sensor (rain gauge or disdrometer)
<code>read_smn(fname)</code>	Reads SwissMetNet data contained in a csv file
<code>read_smn2(fname)</code>	Reads SwissMetNet data contained in a csv file with format station,time,value
<code>read_disdro_scattering(fname)</code>	Reads scattering parameters computed from disdrometer data contained in a text file
<code>read_disdro(fname)</code>	Reads scattering parameters computed from disdrometer data contained in a text file

`pyrad.io.read_data_sensor.get_sensor_data (date, datatype, cfg)`
Gets data from a point measurement sensor (rain gauge or disdrometer)

Parameters

- date** [datetime object] measurement date
- datatype** [str] name of the data type to read
- cfg** [dictionary] dictionary containing sensor information

Returns

- sensordate , sensorvalue, label, period** [tuple] date, value, type of sensor and measurement period

`pyrad.io.read_data_sensor.read_disdro (fname)`
Reads scattering parameters computed from disdrometer data contained in a text file

Parameters

- fname** [str] path of time series file

Returns

date, preciptype, variable, scattering temperature: tuple The read values

`pyrad.io.read_data_sensor.read_disdro_scattering(fname)`

Reads scattering parameters computed from disdrometer data contained in a text file

Parameters

fname [str] path of time series file

Returns

date, preciptype, lwc, rr, zh, zv, zdr, ldr, ah, av, adiff, kdp, deltaco,

rhohv [tuple] The read values

`pyrad.io.read_data_sensor.read_lightning(fname, filter_data=True)`

Reads lightning data contained in a text file. The file has the following fields:

flashnr: (0 is for noise) UTC seconds of the day Time within flash (in seconds) Latitude (decimal degrees) Longitude (decimal degrees) Altitude (m MSL) Power (dBm)

Parameters

fname [str] path of time series file

filter_data [Boolean] if True filter noise (flashnr = 0)

Returns

flashnr, time_data, time_in_flash, lat, lon, alt, dBm [tuple] A tuple containing the read values. None otherwise

`pyrad.io.read_data_sensor.read_lightning_all(fname, labels=['hydro [-]', 'KDPc [deg/Km]', 'dBZc [dBZ]', 'RhoHvc [-]', 'TEMP [deg C]', 'ZDRc [dB]'])`

Reads a file containing lightning data and co-located polarimetric data. fields:

flashnr time data Time within flash (in seconds) Latitude (decimal degrees) Longitude (decimal degrees) Altitude (m MSL) Power (dBm) Polarimetric values at flash position

Parameters

fname [str] path of time series file

labels [list of str] The polarimetric variables labels

Returns

flashnr, time_data, time_in_flash, lat, lon, alt, dBm,

pol_vals_dict [tuple] A tuple containing the read values. None otherwise

`pyrad.io.read_data_sensor.read_lightning_traj(fname)`

Reads lightning trajectory data contained in a csv file. The file has the following fields:

Date UTC [seconds since midnight] # Flash Flash Power (dBm) Value at flash Mean value in a 3x3x3 polar box Min value in a 3x3x3 polar box Max value in a 3x3x3 polar box # valid values in the polar box

Parameters

fname [str] path of time series file

Returns

time_flash, flashnr, dBm, val_at_flash, val_mean, val_min, val_max,
nval [tuple] A tuple containing the read values. None otherwise

`pyrad.io.read_data_sensor.read_meteorage(fname)`

Reads METEORAGE lightning data contained in a text file. The file has the following fields:

date: date + time + time zone lon: longitude [degree] lat: latitude [degree] intens: amplitude [kilo amperes] ns: number of strokes of the flash mode: kind of localization [0,15] intra: 1 = intra-cloud , 0 = cloud-to-ground ax: length of the semi-major axis of the ellipse [km] ki2: standard deviation on the localization computation (Ki^2) ecc: eccentricity (major-axis / minor-axis) incl: ellipse inclination (angle with respect to the North, +90° is

East) [degrees]

sind: stroke index within the flash

Parameters

fname [str] path of time series file

Returns

stroke_time, lon, lat, intens, ns, mode, intra, ax, ki2, ecc, incl,
sind [tuple] A tuple containing the read values. None otherwise

`pyrad.io.read_data_sensor.read_smn(fname)`

Reads SwissMetNet data contained in a csv file

Parameters

fname [str] path of time series file

Returns

smn_id, date , pressure, temp, rh, precip, wspeed, wdir [tuple] The read values

`pyrad.io.read_data_sensor.read_smn2(fname)`

Reads SwissMetNet data contained in a csv file with format station,time,value

Parameters

fname [str] path of time series file

Returns

smn_id, date , value [tuple] The read values

`pyrad.io.read_data_sensor.read_trt_cell_lightning(fname)`

Reads the lightning data of a TRT cell. The file has the following fields:

traj_ID yyyyymmddHHMM lon lat area RANKr nflashes flash_dens

Parameters

fname [str] path of the TRT data file

Returns

A tuple containing the read values. None otherwise

`pyrad.io.read_data_sensor.read_trt_data(fname)`

Reads the TRT data contained in a text file. The file has the following fields:

traj_ID yyyyymmddHHMM

Description of ellipsis: lon [deg] lat [deg] ell_L [km] long ell_S [km] short ell_or [deg] orientation area [km2]

Cell speed: vel_x [km/h] vel_y [km/h] det [dBZ]: detection threshold RANKr from 0 to 40 (int)

Lightning information: CG- number (int) CG+ number (int) CG number (int) %CG+ [%]

Echo top information: ET45 [km] echotop 45 max ET45m [km] echotop 45 median ET15 [km] echotop 15 max ET15m [km] echotop 15 median

VIL and max echo: VIL [kg/m2] vertical integrated liquid content maxH [km] height of maximum reflectivity (maximum on the cell) maxHm [km] height of maximum reflectivity (median per cell)

POH [%] RANK (deprecated)

standard deviation of the current time step cell velocity respect to the previous time: Dvel_x [km/h] Dvel_y [km/h]

cell_contour_lon-lat

Parameters

fname [str] path of the TRT data file

Returns

A tuple containing the read values. None otherwise

`pyrad.io.read_data_sensor.read_trt_scores(fname)`

Reads the TRT scores contained in a text file. The file has the following fields:

traj ID max flash density time max flash density rank max flash density max rank time max rank

Parameters

fname [str] path of the TRT data file

Returns

A tuple containing the read values. None otherwise

`pyrad.io.read_data_sensor.read_trt_traj_data(fname)`

Reads the TRT cell data contained in a text file. The file has the following fields:

traj_ID yyyyymmddHHMM

lon [deg] lat [deg] ell_L [km] long ell_S [km] short ell_or [deg] orientation area [km2]

vel_x [km/h] cell speed vel_y [km/h] det [dBZ] detection threshold RANKr from 0 to 40 (int)

CG- number (int) CG+ number (int) CG number (int) %CG+ [%]

ET45 [km] echotop 45 max ET45m [km] echotop 45 median ET15 [km] echotop 15 max ET15m [km] echotop 15 median VIL [kg/m2] vertical integrated liquid content maxH [km] height of maximum reflectivity (maximum on the cell) maxHm [km] height of maximum reflectivity (median per cell) POH [%] RANK (deprecated)

Standard deviation of the current time step cell velocity respect to the previous time: Dvel_x [km/h] Dvel_y [km/h]

cell_contour_lon-lat

Parameters

fname [str] path of the TRT data file

Returns

A tuple containing the read values. None otherwise

PYRAD.IO.READ_DATA_SUN

Functions for reading data used in sun monitoring

<code>read_sun_hits_multiple_days(cfg, time_ref, ...)</code>	Reads sun hits data from multiple file sources
<code>read_sun_hits(fname)</code>	Reads sun hits data contained in a csv file
<code>read_sun_retrieval(fname)</code>	Reads sun retrieval data contained in a csv file
<code>read_solar_flux(fname)</code>	Reads solar flux data from the DRAO observatory in Canada

`pyrad.io.read_data_sun.read_solar_flux(fname)`
Reads solar flux data from the DRAO observatory in Canada

Parameters

fname [str] path of time series file

Returns

flux_datetime [datetime array] the date and time of the solar flux retrievals

flux_value [array] the observed solar flux

`pyrad.io.read_data_sun.read_sun_hits(fname)`
Reads sun hits data contained in a csv file

Parameters

fname [str] path of time series file

Returns

date, ray, nrng, rad_el, rad_az, sun_el, sun_az, ph, ph_std, npv, nvalh,

pv, pv_std, npv, nvalv, zdr, zdr_std, nzdr, nvalzdr [tuple] Each parameter is an array containing a time series of information on a variable

`pyrad.io.read_data_sun.read_sun_hits_multiple_days(cfg, time_ref, nfiles=1)`
Reads sun hits data from multiple file sources

Parameters

cfg [dict] dictionary with configuration data to find out the right file

time_ref [datetime object] reference time

nfiles [int] number of files to read

Returns

date, ray, nrng, rad_el, rad_az, sun_el, sun_az, ph, ph_std, nph, nvalh,

pv, pv_std, npv, nvalv, zdr, zdr_std, nzdr, nvalzdr [tuple] Each parameter is an array containing a time series of information on a variable

`pyrad.io.read_data_sun.read_sun_retrieval` (*fname*)

Reads sun retrieval data contained in a csv file

Parameters

fname [str] path of time series file

Returns

first_hit_time, last_hit_time, nhits_h, el_width_h, az_width_h, el_bias_h,

az_bias_h, dBm_sun_est, std_dBm_sun_est, sf_h,

nhits_v, el_width_v, az_width_v, el_bias_v, az_bias_v, dBmv_sun_est,

std_dBmv_sun_est, sf_v,

nhits_zdr, zdr_sun_est, std_zdr_sun_est,

sf_ref, ref_time [tuple] Each parameter is an array containing a time series of information on a variable

PYRAD.IO.READ_DATA_OTHER

Functions for reading auxiliary data

<i>read_profile_ts</i> (fname_list, labels[, hres, ...])	Reads a collection of profile data file and creates a time series
<i>read_histogram_ts</i> (fname_list, datatype[, t_res])	Reads a collection of histogram data file and creates a time series
<i>read_quantiles_ts</i> (fname_list[, step, qmin, ...])	Reads a collection of quantiles data file and creates a time series
<i>read_rhi_profile</i> (fname[, labels])	Reads a monitoring time series contained in a csv file
<i>read_last_state</i> (fname)	Reads a file containing the date of acquisition of the last volume processed
<i>read_status</i> (voltime, cfg[, ind_rad])	Reads rad4alp xml status file.
<i>read_rad4alp_cosmo</i> (fname, datatype[, ngates])	Reads rad4alp COSMO data binary file.
<i>read_rad4alp_vis</i> (fname, datatype)	Reads rad4alp visibility data binary file.
<i>read_histogram</i> (fname)	Reads a histogram contained in a csv file
<i>read_quantiles</i> (fname)	Reads quantiles contained in a csv file
<i>read_excess_gates</i> (fname)	Reads a csv files containing the position of gates exceeding a given percentile of frequency of occurrence
<i>read_colocated_gates</i> (fname)	Reads a csv files containing the position of colocated gates
<i>read_colocated_data</i> (fname)	Reads a csv files containing colocated data
<i>read_colocated_data_time_avg</i> (fname)	Reads a csv files containing time averaged colocated data
<i>read_timeseries</i> (fname)	Reads a time series contained in a csv file
<i>read_ts_cum</i> (fname)	Reads a time series of precipitation accumulation contained in a csv file
<i>read_ml_ts</i> (fname)	Reads a melting layer time series contained in a csv file
<i>read_monitoring_ts</i> (fname[, sort_by_date])	Reads a monitoring time series contained in a csv file
<i>read_monitoring_ts_old</i> (fname)	Reads an old format of the monitoring time series contained in a text file
<i>read_intercomp_scores_ts</i> (fname[, sort_by_date])	Reads a radar intercomparison scores csv file
<i>read_intercomp_scores_ts_old</i> (fname)	Reads a radar intercomparison scores csv file in old format
<i>read_intercomp_scores_ts_old_v0</i> (fname[, ...])	Reads a radar intercomparison scores csv file in the oldest format
<i>read_selfconsistency</i> (fname)	Reads a self-consistency table with Zdr, Kdp/Zh columns
<i>read_antenna_pattern</i> (fname[, linear, twoway])	Read antenna pattern from file

`pyrad.io.read_data_other.read_antenna_pattern(fname, linear=False, twoway=False)`

Read antenna pattern from file

Parameters

fname [str] path of the antenna pattern file

linear [boolean] if true the antenna pattern is given in linear units

twoway [boolean] if true the attenuation is two-way

Returns

pattern [dict] dictionary with the fields angle and attenuation

`pyrad.io.read_data_other.read_colocated_data(fname)`

Reads a csv files containing colocated data

Parameters

fname [str] path of time series file

Returns

rad1_time, rad1_ray_ind, rad1_rng_ind, rad1_ele, rad1_azi, rad1_rng,

rad1_val, rad2_time, rad2_ray_ind, rad2_rng_ind, rad2_ele, rad2_azi,

rad2_rng, rad2_val [tuple] A tuple with the data read. None otherwise

`pyrad.io.read_data_other.read_colocated_data_time_avg(fname)`

Reads a csv files containing time averaged colocated data

Parameters

fname [str] path of time series file

Returns

rad1_time, rad1_ray_ind, rad1_rng_ind, rad1_ele , rad1_azi, rad1_rng,

rad1_val, rad2_time, rad2_ray_ind, rad2_rng_ind, rad2_ele, rad2_azi,

rad2_rng, rad2_val [tuple] A tuple with the data read. None otherwise

`pyrad.io.read_data_other.read_colocated_gates(fname)`

Reads a csv files containing the position of colocated gates

Parameters

fname [str] path of time series file

Returns

rad1_ray_ind, rad1_rng_ind, rad1_ele, rad1_azi, rad1_rng,

rad2_ray_ind, rad2_rng_ind, rad2_ele, rad2_azi, rad2_rng [tuple] A tuple with the data read. None otherwise

`pyrad.io.read_data_other.read_excess_gates(fname)`

Reads a csv files containing the position of gates exceeding a given percentile of frequency of occurrence

Parameters

fname [str] path of time series file

Returns

rad1_ray_ind, rad1_rng_ind, rad1_ele, rad1_azi, rad1_rng,

rad2_ray_ind, rad2_rng_ind, rad2_ele, rad2_azl, rad2_rng [tuple] A tuple with the data read. None otherwise

`pyrad.io.read_data_other.read_histogram(fname)`

Reads a histogram contained in a csv file

Parameters

fname [str] path of time series file

Returns

hist, bin_edges [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_histogram_ts(fname_list, datatype, t_res=300.0)`

Reads a collection of histogram data file and creates a time series

Parameters

fname_list [str] list of files to read

datatype [str] The data type (dBZ, ZDR, etc.)

t_res [float] time resolution [s]. If None the time resolution is taken as the median

Returns

tbin_edges, bin_edges, data_ma, datetime_arr[0] [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_intercomp_scores_ts(fname, sort_by_date=False)`

Reads a radar intercomparison scores csv file

Parameters

fname [str] path of time series file

sort_by_date [bool] if True, the read data is sorted by date prior to exit

Returns

date_vec, np_vec, meanbias_vec, medianbias_vec, quant25bias_vec, quant75bias_vec, modebias_vec, corr_vec, slope_vec, intercep_vec, intercep_slope1_vec [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_intercomp_scores_ts_old(fname)`

Reads a radar intercomparison scores csv file in old format

Parameters

fname [str] path of time series file

Returns

date_vec, np_vec, meanbias_vec, medianbias_vec, quant25bias_vec, quant75bias_vec, modebias_vec, corr_vec, slope_vec, intercep_vec, intercep_slope1_vec [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_intercomp_scores_ts_old_v0(fname, corr_min=0.6, np_min=9)`

Reads a radar intercomparison scores csv file in the oldest format

Parameters

fname [str] path of time series file

Returns

**date_vec, np_vec, meanbias_vec, medianbias_vec, quant25bias_vec,
quant75bias_vec, modebias_vec, corr_vec, slope_vec, intercep_vec,
intercep_slope1_vec** [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_last_state(fname)`

Reads a file containing the date of acquisition of the last volume processed

Parameters

fname [str] name of the file to read

Returns

last_state [datetime object] the date

`pyrad.io.read_data_other.read_ml_ts(fname)`

Reads a melting layer time series contained in a csv file

Parameters

fname [str] path of time series file

Returns

**dt_ml, ml_top_avg, ml_top_std, thick_avg, thick_std, nrays_valid,
nrays_total** [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_monitoring_ts(fname, sort_by_date=False)`

Reads a monitoring time series contained in a csv file

Parameters

fname [str] path of time series file

sort_by_date [bool] if True, the read data is sorted by date prior to exit

Returns

date, np_t, central_quantile, low_quantile, high_quantile [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_monitoring_ts_old(fname)`

Reads an old format of the monitoring time series contained in a text file

Parameters

fname [str] path of time series file

Returns

date, np_t, central_quantile, low_quantile, high_quantile [tuple] The read data in the current format. None otherwise

`pyrad.io.read_data_other.read_profile_ts(fname_list, labels, hres=None, label_nr=0,
t_res=300.0)`

Reads a collection of profile data file and creates a time series

Parameters

fname_list [str] list of files to read

labels [list of str] The data labels

hres [float] Height resolution

label_nr [int] the label nr of the data that will be used in the time series

t_res [float] time resolution [s]. If None the time resolution is taken as the median

Returns

tbin_edges, hbin_edges, np_ma, data_ma, datetime_arr[0] [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_quantiles(fname)`

Reads quantiles contained in a csv file

Parameters

fname [str] path of time series file

Returns

quantiles, values [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_quantiles_ts(fname_list, step=5.0, qmin=0.0, qmax=100.0, t_res=300.0)`

Reads a collection of quantiles data file and creates a time series

Parameters

fname_list [str] list of files to read

step, qmin, qmax [float] The minimum, maximum and step quantiles

t_res [float] time resolution [s]. If None the time resolution is taken as the median

Returns

tbin_edges, qbin_edges, data_ma, datetime_arr[0] [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_rad4alp_cosmo(fname, datatype, ngates=0)`

Reads rad4alp COSMO data binary file.

Parameters

fname [str] name of the file to read

datatype [str] name of the data type

ngates [int] maximum number of range gates per ray. If larger than 0 the radar field will be cut accordingly.

Returns

field [dictionary] The data field

`pyrad.io.read_data_other.read_rad4alp_vis(fname, datatype)`

Reads rad4alp visibility data binary file.

Parameters

fname [str] name of the file to read

datatype [str] name of the data type

Returns

field_list [list of dictionaries] A data field. Each element of the list corresponds to one elevation

`pyrad.io.read_data_other.read_rhi_profile(fname, labels=['50.0-percentile', '25.0-percentile', '75.0-percentile'])`

Reads a monitoring time series contained in a csv file

Parameters

fname [str] path of time series file

labels [list of str] The data labels

Returns

height, np_t, vals [tuple] The read data. None otherwise

`pyrad.io.read_data_other.read_selfconsistency(fname)`
Reads a self-consistency table with Zdr, Kdp/Zh columns

Parameters

fname [str] path of time series file

Returns

zdr, kdpzh [arrays] The read values

`pyrad.io.read_data_other.read_status(voltime, cfg, ind_rad=0)`
Reads rad4alp xml status file.

Parameters

voltime [datetime object] volume scan time

cfg: dictionary of dictionaries configuration info to figure out where the data is

ind_rad: int radar index

Returns

root [root element object] The information contained in the status file

`pyrad.io.read_data_other.read_timeseries(fname)`
Reads a time series contained in a csv file

Parameters

fname [str] path of time series file

Returns

date, value [tuple] A datetime object array containing the time and a numpy masked array containing the value. None otherwise

`pyrad.io.read_data_other.read_ts_cum(fname)`
Reads a time series of precipitation accumulation contained in a csv file

Parameters

fname [str] path of time series file

Returns

date, np_radar, radar_value, np_sensor, sensor_value [tuple] The data read

PYRAD.IO.WRITE_DATA

Functions for writing pyrad output data

<code>write_ts_lightning(flashnr, time_data, ...)</code>	writes the LMA sources data and the value of the collocated polarimetric variables
<code>send_msg(sender, receiver_list, subject, fname)</code>	sends the content of a text file by email
<code>write_alarm_msg(radar_name, param_name_unit, ...)</code>	writes an alarm file
<code>write_last_state(datetime_last, fname)</code>	writes SwissMetNet data in format datetime,avg_value, std_value
<code>write_smn(datetime_vec, value_avg_vec, ...)</code>	writes SwissMetNet data in format datetime,avg_value, std_value
<code>write_trt_cell_data(traj_ID, yyyymmddHHMM, ...)</code>	writes TRT cell data
<code>write_trt_cell_scores(traj_ID, ...)</code>	writes TRT cells scores
<code>write_trt_cell_lightning(cell_ID, cell_time, ...)</code>	writes the lightning data for each TRT cell
<code>write_rhi_profile(hvec, data, nvalid_vec, ...)</code>	writes the values of an RHI profile in a text file
<code>write_field_coverage(quantiles, values, ...)</code>	writes the quantiles of the coverage on a particular sector
<code>write_cdf(quantiles, values, ntot, nnan, ...)</code>	writes a cumulative distribution function
<code>write_histogram(bin_edges, values, fname[, ...])</code>	writes a histogram
<code>write_quantiles(quantiles, values, fname[, ...])</code>	writes quantiles
<code>write_ts_polar_data(dataset, fname)</code>	writes time series of data
<code>write_ts_ml(dt_ml, ml_top_avg, ml_top_std, ...)</code>	writes time series of melting layer data
<code>write_ts_cum(dataset, fname)</code>	writes time series accumulation of data
<code>write_monitoring_ts(start_time, np_t, ...[, ...])</code>	writes time series of data
<code>write_excess_gates(excess_dict, fname)</code>	Writes the position and values of gates that have a frequency of occurrence higher than a particular threshold
<code>write_intercomp_scores_ts(start_time, stats, ...)</code>	writes time series of radar intercomparison scores
<code>write_colocated_gates(coloc_gates, fname)</code>	Writes the position of gates colocated with two radars
<code>write_colocated_data(coloc_data, fname)</code>	Writes the data of gates colocated with two radars
<code>write_colocated_data_time_avg(coloc_data, fname)</code>	Writes the time averaged data of gates colocated with two radars
<code>write_sun_hits(sun_hits, fname)</code>	Writes sun hits data.
<code>write_sun_retrieval(sun_retrieval, fname)</code>	Writes sun retrieval data.

`pyrad.io.write_data.send_msg(sender, receiver_list, subject, fname)`
sends the content of a text file by email

Parameters

sender [str] the email address of the sender

receiver_list [list of string] list with the email addresses of the receiver

subject [str] the subject of the email

fname [str] name of the file containing the content of the email message

Returns

fname [str] the name of the file containing the content

```
pyrad.io.write_data.write_alarm_msg(radar_name, param_name_unit, date_last, target,  
                                     tol_abs, np_trend, value_trend, tol_trend, nevents,  
                                     np_last, value_last, fname)
```

writes an alarm file

Parameters

radar_name [str] Name of the radar being controlled

param_name_unit [str] Parameter and units

date_last [datetime object] date of the current event

target, tol_abs [float] Target value and tolerance

np_trend [int] Total number of points in trend

value_trend, tol_trend [float] Trend value and tolerance

nevents: int Number of events in trend

np_last [int] Number of points in the current event

value_last [float] Value of the current event

fname [str] Name of file where to store the alarm information

Returns

fname [str] the name of the file where data has written

```
pyrad.io.write_data.write_cdf(quantiles, values, ntot, nnan, nclut, nblocked, nprec_filter, nout-  
                             liers, ncdf, fname, use_nans=False, nan_value=0.0, filterprec=[],  
                             vismin=None, sector=None, datatype=None, timeinfo=None)
```

writes a cumulative distribution function

Parameters

quantiles [datetime array] array containing the measurement time

values [float array] array containing the average value

fname [float array] array containing the standard deviation

sector [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

```
pyrad.io.write_data.write_colocated_data(coloc_data, fname)
```

Writes the data of gates colocated with two radars

Parameters

coloc_data [dict] dictionary containing the colocated data parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_colocated_data_time_avg(coloc_data, fname)`

Writes the time averaged data of gates colocated with two radars

Parameters

coloc_data [dict] dictionary containing the colocated data parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_colocated_gates(coloc_gates, fname)`

Writes the position of gates colocated with two radars

Parameters

coloc_gates [dict] dictionary containing the colocated gates parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_excess_gates(excess_dict, fname)`

Writes the position and values of gates that have a frequency of occurrence higher than a particular threshold

Parameters

excess_dict [dict] dictionary containing the gates parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_field_coverage(quantiles, values, ele_start, ele_stop, azi_start,
azi_stop, threshold, nvalid_min, datatype, timeinfo, fname)`

writes the quantiles of the coverage on a particular sector

Parameters

quantiles [datetime array] array containing the quantiles computed

values [float array] quantile value

ele_start, ele_stop, azi_start, azi_stop [float] The limits of the sector

threshold [float] The minimum value to consider the data valid

nvalid_min [int] the minimum number of points to consider that there are values in a ray

datatype [str] data type and units

timeinfo [datetime object] the time stamp of the data

fname [str] name of the file where to write the data

Returns

fname [str] the name of the file where data has written

```
pyrad.io.write_data.write_histogram(bin_edges, values, fname, datatype='undefined',  
                                     step=0)
```

writes a histogram

Parameters

bin_edges [float array] array containing the histogram bin edges

values [int array] array containing the number of points in each bin

fname [str] file name

datatype :str The data type

step [str] The bin step

Returns

fname [str] the name of the file where data has written

```
pyrad.io.write_data.write_intercomp_scores_ts(start_time, stats, field_name,  
                                              fname, rad1_name='RADAR001',  
                                              rad2_name='RADAR002',  
                                              rewrite=False)
```

writes time series of radar intercomparison scores

Parameters

start_time [datetime object or array of date time objects] the time of the intercomparison

stats [dict] dictionary containing the statistics

field_name [str] The name of the field

fname [str] file name where to store the data

rad1_name, rad2_name [str] Name of the radars intercompared

rewrite [bool] if True a new file is created

Returns

fname [str] the name of the file where data has written

```
pyrad.io.write_data.write_last_state(datetime_last, fname)  
writes SwissMetNet data in format datetime, avg_value, std_value
```

Parameters

datetime_last [datetime object] date and time of the last state

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

```
pyrad.io.write_data.write_monitoring_ts(start_time, np_t, values, quantiles, datatype, fname,  
                                         rewrite=False)
```

writes time series of data

Parameters

start_time [datetime object or array of date time objects] the time of the monitoring

np_t [int or array of ints] the total number of points

values: float array with 3 elements of array of arrays the values at certain quantiles

quantiles: float array with 3 elements the quantiles computed

datatype [str] The data type

fname [str] file name where to store the data

rewrite [bool] if True a new file is created

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_quantiles` (*quantiles, values, fname, datatype='undefined'*)
writes quantiles

Parameters

quantiles [float array] array containing the quantiles to write

values [float array] array containing the value of each quantile

fname [str] file name

datatype :str The data type

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_rhi_profile` (*hvec, data, nvalid_vec, labels, fname, datatype=None, timeinfo=None, sector=None*)

writes the values of an RHI profile in a text file

Parameters

hvec [float array] array containing the altitude in m MSL

data [list of float array] the quantities at each altitude

nvalid_vec [int array] number of valid data points used to compute the quantiles

labels [list of strings] label specifying the quantities in data

fname [str] file name where to store the data

datatype [str] the data type

timeinfo [datetime object] time of the rhi profile

sector [dict] dictionary specifying the sector limits

Returns

fname [str] the name of the file where data has been written

`pyrad.io.write_data.write_smn` (*datetime_vec, value_avg_vec, value_std_vec, fname*)
writes SwissMetNet data in format datetime, avg_value, std_value

Parameters

datetime_vec [datetime array] array containing the measurement time

value_avg_vec [float array] array containing the average value

value_std_vec [float array] array containing the standard deviation

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_sun_hits(sun_hits, fname)`

Writes sun hits data.

Parameters

sun_hits [dict] dictionary containing the sun hits parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_sun_retrieval(sun_retrieval, fname)`

Writes sun retrieval data.

Parameters

sun_retrieval [dict] dictionary containing the sun retrieval parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_trt_cell_data(traj_ID, yyyymmddHHMM, lon, lat, ell_L, ell_S,
ell_or, area, vel_x, vel_y, det, RANKr, CG_n,
CG_p, CG, CG_percent_p, ET45, ET45m, ET15,
ET15m, VIL, maxH, maxHm, POH, RANK, Dvel_x,
Dvel_y, cell_contour, fname)`

writes TRT cell data

Parameters

traj_ID, **yyyymmddHHMM**, **lon**, **lat**, **ell_L**, **ell_S**, **ell_or**, **area**,
vel_x, **vel_y**, **det**, **RANKr**, **CG_n**, **CG_p**, **CG**, **CG_percent_p**, **ET45**,
ET45m, **ET15**, **ET15m**, **VIL**, **maxH**, **maxHm**, **POH**, **RANK**, **Dvel_x**,

Dvel_y, **cell_contour**: the cell parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_trt_cell_lightning(cell_ID, cell_time, lon, lat, area, rank,
nflash, flash_density, fname)`

writes the lightning data for each TRT cell

Parameters

cell_ID [array of ints] the cell ID

cell_time [array of datetime] the time step

lon, **lat** [array of floats] the latitude and longitude of the center of the cell

area [array of floats] the area of the cell

rank [array of floats] the rank of the cell

nflash [array of ints] the number of flashes/sources within the cell

flash_density [array of floats] the flash/source density

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

```
pyrad.io.write_data.write_trt_cell_scores (traj_ID, flash_density_max_time,  
                                             flash_density_max_rank, nflashes_max_list,  
                                             area_flash_max_list, flash_density_max,  
                                             rank_max_time, rank_max, fname)
```

writes TRT cells scores

Parameters

traj_ID [array of ints] The ID of the cells

flash_density_max_time [array of date times] The time at which the maximum flash density was reached for each cell

flash_density_max_rank [array of floats] The rank when the maximum flash density was reached for each cell

nflashes_max_list [array of ints] the number of flashes when the max flash density was reached

area_flash_max_list [array of floats] The area when the max flash density was reached

flash_density_max [array of floats] The maximum flash density for each cell

rank_max_time [array of datetime] the time at wich the maximum rank of each cell was reached

rank_max [array of float] the rank when the maximum rank of each cell was reached

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

```
pyrad.io.write_data.write_ts_cum (dataset, fname)
```

writes time series accumulation of data

Parameters

dataset [dict] dictionary containing the time series parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

```
pyrad.io.write_data.write_ts_lightning (flashnr, time_data, time_in_flash, lat, lon, alt, dBm,  
                                         vals_list, fname, pol_vals_labels)
```

writes the LMA sources data and the value of the colocated polarimetric variables

Parameters

flashnr [int] flash number

time_data [datetime object] flash source time

time_in_flash [float] seconds since start of flash

lat, lon, alt [float] latitude, longitude [deg] and altitude [m MSL] of the flash source

dBm [float] flash power

vals_list [list of arrays] List containing the data for each polarimetric variable

fname [str] the name of the file containing the content

pol_values_labels [list of strings] List containing strings identifying each polarimetric variable

Returns

fname [str] the name of the file containing the content

`pyrad.io.write_data.write_ts_ml(dt_ml, ml_top_avg, ml_top_std, thick_avg, thick_std, nrays_valid, nrays_total, fname)`

writes time series of melting layer data

Parameters

dt_ml [date time array] array of time steps

ml_top_avg, ml_top_std: float arrays the average and the standard deviation of the melting layer top height

thick_avg, thick_std: float arrays the average and the standard deviation of the metling layer thickness

nrays_valid, nrays_total: int arrays the number of rays where melting layer has been identified and the total number of arrays in the scan

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

`pyrad.io.write_data.write_ts_polar_data(dataset, fname)`

writes time series of data

Parameters

dataset [dict] dictionary containing the time series parameters

fname [str] file name where to store the data

Returns

fname [str] the name of the file where data has written

PYRAD.IO.TIMESERIES

TimeSeries class implementation for holding timeseries data.

<i>TimeSeries</i> (desc[, timevec, timeformat, ...])	Holding timeseries data and metadata.
--	---------------------------------------

```
class pyrad.io.timeseries.TimeSeries (desc,          timevec=None,          timeformat=None,
                                     maxlen=None, datatype="")
```

Bases: `object`

Holding timeseries data and metadata.

Attributes

description [array of str] Description of the data of the time series.

time_vector [array of datetime objects]

timeformat [how to print the time (default:)] 'Date, UTC [seconds since midnight]'

dataseries [List of _dataSeries object holding the] data

Methods

<i>add_dataseries</i> (label, unit_name, unit[, ...])	Add a new data series to the timeseries object.
<i>add_timesample</i> (dt, values)	Add a new sample to the time series.
<i>plot</i> (fname[, ymin, ymax])	Make a figure of a time series
<i>plot_hist</i> (fname[, step])	Make histograms of time series
<i>write</i> (fname)	Write time series output

__class__

alias of `builtins.type`

__delattr__ (*\$self, name, /*)

Implement `delattr(self, name)`.

__dict__ = `mappingproxy({'__module__': 'pyrad.io.timeseries', '__doc__': "\n Holding`

__dir__ () → list

default `dir()` implementation

__eq__ (*\$self, value, /*)

Return `self==value`.

__format__ ()

default object formatter

__ge__ (*\$self, value, /*)
Return self>=value.

__getattr__ (*\$self, name, /*)
Return getattr(self, name).

__gt__ (*\$self, value, /*)
Return self>value.

__hash__ (*\$self, /*)
Return hash(self).

__init__ (*desc, timevec=None, timeformat=None, maxlength=None, datatype=""*)
Initialize the object.

Parameters

desc [array of str]
timevec [array of datetime]
timeformat [specifies time format]
maxlength [Maximal length of the time series]
num_el [Number of values in the time series]

__init_subclass__ ()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__ (*\$self, value, /*)
Return self<=value.

__lt__ (*\$self, value, /*)
Return self<value.

__module__ = 'pyrad.io.timeseries'

__ne__ (*\$self, value, /*)
Return self!=value.

__new__ (*\$type, *args, **kwargs*)
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__ (*\$self, /*)
Return repr(self).

__setattr__ (*\$self, name, value, /*)
Implement setattr(self, name, value).

__sizeof__ () → int
size of object in memory, in bytes

__str__ (*\$self, /*)
Return str(self).

__subclasshook__ ()

Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__

list of weak references to the object (if defined)

add_dataserries (*label, unit_name, unit, dataserries=None, plot=True, color=None, linestyle=None*)

Add a new data series to the timeseries object. The length of the data vector must be the same as the length of the time vector.

add_timesample (*dt, values*)

Add a new sample to the time series.

plot (*fname, ymin=None, ymax=None*)

Make a figure of a time series

plot_hist (*fname, step=None*)

Make histograms of time series

write (*fname*)

Write time series output

class pyrad.io.timeseries._DataSeries (*label, unit_name, unit, data, plot=True, color=None, linestyle=None*)

Bases: `object`

Hold a data vector and some meta information.

Methods

set_value(*i, val*)

Append value to array

__class__

alias of `builtins.type`

__delattr__ (*\$self, name, /*)

Implement `delattr(self, name)`.

__dict__ = `mappingproxy({'__module__': 'pyrad.io.timeseries', '__doc__': '\n Hold a c`

__dir__ () → list

default `dir()` implementation

__eq__ (*\$self, value, /*)

Return `self==value`.

__format__ ()

default object formatter

__ge__ (*\$self, value, /*)

Return `self>=value`.

__getattr__ (*\$self, name, /*)

Return `getattr(self, name)`.

__gt__ (*\$self, value, /*)

Return `self>value`.

__hash__ (\$self, /)
Return hash(self).

__init__ (label, unit_name, unit, data, plot=True, color=None, linestyle=None)
Initialize the object.

__init_subclass__ ()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__ (\$self, value, /)
Return self<=value.

__lt__ (\$self, value, /)
Return self<value.

__module__ = 'pyrad.io.timeseries'

__ne__ (\$self, value, /)
Return self!=value.

__new__ (\$type, *args, **kwargs)
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__ (\$self, /)
Return repr(self).

__setattr__ (\$self, name, value, /)
Implement setattr(self, name, value).

__sizeof__ () → int
size of object in memory, in bytes

__str__ (\$self, /)
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

set_value (i, val)
Append value to array

PYRAD.IO.TRAJECTORY

Trajectory class implementation for reading trajectory file. Converting to different coordinate systems.

<code>Trajectory(filename[, starttime, endtime, ...])</code>	A class for reading and handling trajectory data from a file.
<code>_Radar_Trajectory(lat, lon, alt)</code>	A class for holding the trajectory data assigned to a radar.

```
class pyrad.io.trajectory.Trajectory (filename,  starttime=None,  endtime=None,  traj-  
                                         type='plane', flashnr=0)
```

Bases: `object`

A class for reading and handling trajectory data from a file.

Attributes

filename [str] Path and name of the trajectory definition file

starttime [datetime] Start time of trajectory processing.

endtime [datetime] End time of trajectory processing.

trajtype [str]

Type of trajectory. Can be 'plane' or 'lightning'

time_vector [Array of datetime objects] Array containing the trajectory time samples

wgs84_lat_deg [Array of floats] WGS84 latitude samples in radian

wgs84_lon_deg [Array of floats] WGS84 longitude samples in radian

wgs84_alt_m [Array of floats] WGS84 altitude samples in m

nsamples [int]

Number of samples in the trajectory

_swiss_grid_done [Bool] Indicates that conversion to Swiss coordinates has been performed

swiss_chy, swiss_chx, swiss_chh [Array of floats] Swiss coordinates in m

radar_list [list] List of radars for which trajectories are going to be computed

flashnr [int] For 'lightning' only. Number of flash for which trajectory data is going to be computed. If 0 all all flashes are going to be considered.

time_in_flash [array of floats] For 'lightning' only. Time within flash (sec)

flashnr_vec [array of ints] For 'lightning' only. Flash number of each data sample

dBm [array of floats] For 'lightning' only. Lightning power (dBm)

Methods

<code>add_radar(radar)</code>	Add the coordinates (WGS84 longitude, latitude and non WGS84 altitude) of a radar to the radar_list.
<code>calculate_velocities(radar)</code>	Calculate velocities.
<code>get_end_time()</code>	Get time of last trajectory sample.
<code>get_samples_in_period([start, end])</code>	”
<code>get_start_time()</code>	Get time of first trajectory sample.

`__class__`

alias of `builtins.type`

`__delattr__ ($self, name, /)`

Implement `delattr(self, name)`.

`__dict__ = mappingproxy({'__module__': 'pyrad.io.trajectory', '__doc__': "\n A class`

`__dir__ () → list`

default `dir()` implementation

`__eq__ ($self, value, /)`

Return `self==value`.

`__format__ ()`

default object formatter

`__ge__ ($self, value, /)`

Return `self>=value`.

`__getattr__ ($self, name, /)`

Return `getattr(self, name)`.

`__gt__ ($self, value, /)`

Return `self>value`.

`__hash__ ($self, /)`

Return `hash(self)`.

`__init__ (filename, starttime=None, endtime=None, trajtype='plane', flashnr=0)`

Initialize the object.

Parameters

filename [str] Filename containing the trajectory samples.

starttime [datetime] Start time of trajectory processing. If not given, use the time of the first trajectory sample.

endtime [datetime] End time of trajectory processing. If not given, use the time of the last trajectory sample.

trajtype [str] type of trajectory. Can be plane or lightning

flashnr [int] If type of trajectory is lightning, the flash number to check the trajectory. 0 means all flash numbers included

`__init_subclass__ ()`

This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__ (\$self, value, /)
Return self<=value.

__lt__ (\$self, value, /)
Return self<value.

__module__ = 'pyrad.io.trajectory'

__ne__ (\$self, value, /)
Return self!=value.

__new__ (\$type, *args, **kwargs)
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__ (\$self, /)
Return repr(self).

__setattr__ (\$self, name, value, /)
Implement setattr(self, name, value).

__sizeof__ () → int
size of object in memory, in bytes

__str__ (\$self, /)
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

_convert_traj_to_swissgrid ()
Convert trajectory samples from WGS84 to Swiss CH1903 coordinates

_get_total_seconds (x)
Return total seconds of timedelta object

_read_traj ()
Read trajectory from file

_read_traj_lightning (flashnr=0)
Read trajectory from lightning file

Parameters

flashnr [int] the flash number to keep. If 0 data from all flashes will be kept

_read_traj_trt ()
Read trajectory from TRT file

add_radar (radar)
Add the coordinates (WGS84 longitude, latitude and non WGS84 altitude) of a radar to the radar_list.

Parameters

radar [pyart radar object] containing the radar coordinates

calculate_velocities (*radar*)

Calculate velocities.

get_end_time ()

Get time of last trajectory sample.

get_samples_in_period (*start=None, end=None*)

” Get indices of samples of the trajectory within given time period.

get_start_time ()

Get time of first trajectory sample.

class pyrad.io.trajectory._Radar_Trajectory (*lat, lon, alt*)

Bases: `object`

A class for holding the trajectory data assigned to a radar.

Attributes

latitude [float] WGS84 radar latitude [deg]

longitude [float] WGS84 radar longitude [deg]

altitude [float] radar altitude [m] (non WGS84)

ch_y, ch_x, ch_alt [float] radar coordinates in swiss CH1903 coordinates

elevation_vec [float list] Elevation values of the trajectory samples

azimuth_vec [float list] Azimuth values of the trajectory samples

range_vec [float list] Range values of the trajectory samples

v_abs, v_r, v_el, v_az [array-like] Velocity vectors of the absolute [m/s], radial [m/s], elevation [deg/s] and azimuth [deg/s] velocities

Methods

<code>assign_trajectory</code> (el, az, rr)	Assign a trajectory to the radar in polar radar coordinates.
<code>assign_velocity_vecs</code> (v_abs, v_r, v_el, v_az)	Assign velocity vectors to the radar.
<code>convert_radpos_to_swissgrid</code> ()	Convert the radar location (in WGS84 coordinates) to swiss CH1903 coordinates.
<code>location_is_equal</code> (lat, lon, alt)	Check if the given coordinates are the same.

__class__

alias of `builtins.type`

__delattr__ (*\$self, name, /*)

Implement `delattr(self, name)`.

__dict__ = `mappingproxy({'__module__': 'pyrad.io.trajectory', '__doc__': '\n A class`

__dir__ () → list

default `dir()` implementation

__eq__ (*\$self, value, /*)

Return `self==value`.

__format__ ()
default object formatter

__ge__ (\$self, value, /)
Return self>=value.

__getattr__ (\$self, name, /)
Return getattr(self, name).

__gt__ (\$self, value, /)
Return self>value.

__hash__ (\$self, /)
Return hash(self).

__init__ (lat, lon, alt)
Initialize the object.

Parameters

lat, lon, alt [radar location coordinates]

nsamps [number of samples]

__init_subclass__ ()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__le__ (\$self, value, /)
Return self<=value.

__lt__ (\$self, value, /)
Return self<value.

__module__ = 'pyrad.io.trajectory'

__ne__ (\$self, value, /)
Return self!=value.

__new__ (\$type, *args, **kwargs)
Create and return a new object. See help(type) for accurate signature.

__reduce__ ()
helper for pickle

__reduce_ex__ ()
helper for pickle

__repr__ (\$self, /)
Return repr(self).

__setattr__ (\$self, name, value, /)
Implement setattr(self, name, value).

__sizeof__ () → int
size of object in memory, in bytes

__str__ (\$self, /)
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by `abc.ABCMeta.__subclasscheck__()`. It should return `True`, `False` or `NotImplemented`. If it returns `NotImplemented`, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

`__weakref__`

list of weak references to the object (if defined)

`assign_trajectory` (*el, az, rr*)

Assign a trajectory to the radar in polar radar coordinates.

Parameters

el, az, rr [array-like] elevation, azimuth and range vector

`assign_velocity_vecs` (*v_abs, v_r, v_el, v_az*)

Assign velocity vectors to the radar.

`convert_radpos_to_swissgrid` ()

Convert the radar location (in WGS84 coordinates) to swiss CH1903 coordinates.

`location_is_equal` (*lat, lon, alt*)

Check if the given coordinates are the same.

Parameters

lat, lon, alt [radar location coordinates]

PYRAD.GRAPH.PLOTS_AUX

Auxiliary plotting functions

<code>get_colobar_label(field_dict, field_name)</code>	creates the colorbar label using field metadata
<code>get_field_name(field_dict, field)</code>	Return a nice field name for a particular field
<code>get_norm(field_name)</code>	Computes the normalization of the colormap, and gets the ticks and labels of the colorbar from the metadata of the field.

`pyrad.graph.plots_aux.get_colobar_label (field_dict, field_name)`
creates the colorbar label using field metadata

Parameters

field_dict [dict] dictionary containing field metadata

field_name [str] name of the field

Returns

label [str] colorbar label

`pyrad.graph.plots_aux.get_field_name (field_dict, field)`
Return a nice field name for a particular field

Parameters

field_dict [dict] dictionary containing field metadata

field [str] name of the field

Returns

field_name [str] the field name

`pyrad.graph.plots_aux.get_norm (field_name)`

Computes the normalization of the colormap, and gets the ticks and labels of the colorbar from the metadata of the field. Returns None if the required parameters are not present in the metadata

Parameters

field_name [str] name of the field

Returns

norm [list] the colormap index

ticks [list] the list of ticks in the colorbar

labels [list] the list of labels corresponding to each tick

PYRAD.GRAPH.PLOTS

Functions to plot Pyrad datasets

<code>plot_density(hist_obj, hist_type, ...[,...])</code>	density plot (angle-values representation)
<code>plot_scatter(bin_edges1, bin_edges2, ...[,...])</code>	2D histogram
<code>plot_quantiles(quant, value, fname_list[,...])</code>	plots quantiles
<code>plot_histogram(bin_edges, values, fname_list)</code>	computes and plots histogram
<code>plot_histogram2(bin_centers, hist, fname_list)</code>	plots histogram
<code>plot_antenna_pattern(antpattern, fname_list)</code>	plots an antenna pattern
<code>plot_scatter_comp(value1, value2, fname_list)</code>	plots the scatter between two time series
<code>plot_sun_hits(field, field_name, fname_list, ...)</code>	plots the sun hits

`pyrad.graph.plots.plot_antenna_pattern` (*antpattern*, *fname_list*, *labelx*='Angle [Deg]', *linear*=*False*, *twoway*=*False*, *title*='Antenna Pattern', *ymin*=*None*, *ymax*=*None*, *dpi*=72)

plots an antenna pattern

Parameters

antpattern [dict] dictionary with the angle and the attenuation

value [float array] values of the time series

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

linear [boolean] if true data is in linear units

linear [boolean] if true data represents the two way attenuation

titl [str] The figure title

ymin, ymax: float Lower/Upper limit of y axis

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots.plot_density` (*hist_obj*, *hist_type*, *field_name*, *ind_sweep*, *prdcfg*, *fname_list*, *quantiles*=[25.0, 50.0, 75.0], *ref_value*=0.0, *vmin*=*None*, *vmax*=*None*)

density plot (angle-values representation)

Parameters

hist_obj [histogram object] object containing the histogram data to plot

hist_type [str] type of histogram (instantaneous data or cumulative)

field_name [str] name of the radar field to plot

ind_sweep [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

quantiles [array] the quantile lines to plot

ref_value [float] the reference value

vmin, vmax [float] Minim and maximum extend of the vertical axis

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots.plot_histogram(bin_edges, values, fname_list, labelx='bins', labely='Number of Samples', titl='histogram', dpi=72)`
computes and plots histogram

Parameters

bin_edges [array] histogram bin edges

values [array] data values

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

titl [str] The figure title

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots.plot_histogram2(bin_centers, hist, fname_list, labelx='bins', labely='Number of Samples', titl='histogram', dpi=72, ax=None, fig=None, save_fig=True, color=None, alpha=None, invert_xaxis=False)`

plots histogram

Parameters

bin_centers [array] histogram bin centers

hist [array] values for each bin

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

titl [str] The figure title

dpi [int] dots per inch

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

color [str] color of the bars

alpha [float] parameter controlling the transparency

invert_xaxis [bool] If true inverts the x axis

Returns

fname_list or **fig, ax: list of str** list of names of the created plots

```
pyrad.graph.plots.plot_quantiles(quant, value, fname_list, labelx='quantile', labely='value',  
                                titl='quantile', vmin=None, vmax=None, dpi=72)
```

plots quantiles

Parameters

quant [array] quantiles to be plotted

value [array] values of each quantile

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

titl [str] The figure title

vmin, vmax: float Lower/Upper limit of data values

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

```
pyrad.graph.plots.plot_scatter(bin_edges1, bin_edges2, hist_2d, field_name1, field_name2,  
                               fname_list, prdcfg, metadata=None, lin_regr=None,  
                               lin_regr_slope1=None, rad1_name='RADAR001',  
                               rad2_name='RADAR002')
```

2D histogram

Parameters

bin_edges1, bin_edges2 [float array2] the bins of each field

hist_2d [ndarray 2D] the 2D histogram

field_name1, field_name2 [str] the names of each field

fname_list [list of str] list of names of the files where to store the plot

prdcfg [dict] product configuration dictionary

metadata [str] a string with metadata to write in the plot

lin_regr [tuple with 2 values] the coefficients for a linear regression

lin_regr_slope1 [float] the intercep point of a linear regression of slope 1

rad1_name, rad2_name [str] name of the radars which data is used

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots.plot_scatter_comp` (*value1*, *value2*, *fname_list*, *labelx*='Sensor 1', *labely*='Sensor 2', *titl*='Scatter', *axis*=None, *metadata*=None, *dpi*=72)

plots the scatter between two time series

Parameters

value1 [float array] values of the first time series
value2 [float array] values of the second time series
fname_list [list of str] list of names of the files where to store the plot
labelx [str] The label of the X axis
labely [str] The label of the Y axis
titl [str] The figure title
axis [str] type of axis
metadata [string] a string containing metadata
dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots.plot_sun_hits` (*field*, *field_name*, *fname_list*, *prdcfg*)
plots the sun hits

Parameters

radar [Radar object] object containing the radar data to plot
field_name [str] name of the radar field to plot
altitude [float] the altitude [m MSL] to be plotted
prdcfg [dict] dictionary containing the product configuration
fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

PYRAD.GRAPH.PLOTS_VOL

Functions to plot radar volume data

<code>plot_ppi(radar, field_name, ind_el, prdcfg, ...)</code>	plots a PPI
<code>plot_ppi_map(radar, field_name, ind_el, ...)</code>	plots a PPI on a geographic map
<code>plot_rhi(radar, field_name, ind_az, prdcfg, ...)</code>	plots an RHI
<code>plot_bscope(radar, field_name, ind_sweep, ...)</code>	plots a B-Scope (angle-range representation)
<code>plot_time_range(radar, field_name, ...)</code>	plots a time-range plot
<code>plot_cappi(radar, field_name, altitude, ...)</code>	plots a Constant Altitude Plan Position Indicator CAPPI
<code>plot_traj(rng_traj, azi_traj, ele_traj, ...)</code>	plots a trajectory on a Cartesian surface
<code>plot_rhi_contour(radar, field_name, ind_az, ...)</code>	plots contour data on an RHI
<code>plot_ppi_contour(radar, field_name, ind_el, ...)</code>	plots contour data on a PPI
<code>plot_pos(lat, lon, alt, fname_list[, ax, ...])</code>	plots a trajectory on a Cartesian surface
<code>plot_rhi_profile(data_list, hvec, fname_list)</code>	plots an RHI profile
<code>plot_along_coord(xval_list, yval_list, ...)</code>	plots a time series
<code>plot_field_coverage(xval_list, yval_list, ...)</code>	plots a time series
<code>_plot_time_range(rad_time, rad_range, ...[, ...])</code>	plots a time-range plot

`pyrad.graph.plots_vol._plot_time_range(rad_time, rad_range, rad_data, field_name, fname_list, titl='Time-Range plot', xlabel='time (s from start time)', ylabel='range (Km)', clabel=None, vmin=None, vmax=None, figsize=[10, 8], dpi=72)`

plots a time-range plot

Parameters

rad_time [Radar object] object containing the radar data to plot

rad_range [str] name of the radar field to plot

rad_data [int] sweep index to plot

field_name [str or None] field name. Used to define plot characteristics

fname_list [list of str] list of names of the files where to store the plot

titl [str] Plot title

xlabel, ylabel [str] x- and y-axis labels

clabel [str or None] colorbar label

vmin, vmax [float] min and max values of the color bar

figsize [list] figure size [xsize, ysize]

dpi [int] dpi

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_vol.plot_along_coord(xval_list, yval_list, fname_list, labelx='coord', labely='Value', labels=None, title='Plot along coordinate', colors=None, linestyle=None, ymin=None, ymax=None, dpi=72)`

plots a time series

Parameters

xval_list [list of float arrays] the x values, range, azimuth or elevation

yval_list [list of float arrays] the y values. Parameter to plot

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

labels [array of str] The label of the legend

title [str] The figure title

colors [array of str] Specifies the colors of each line

linestyles [array of str] Specifies the line style of each line

ymin, ymax: float Lower/Upper limit of y axis

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_vol.plot_bscope(radar, field_name, ind_sweep, prdcfg, fname_list)`
plots a B-Scope (angle-range representation)

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_sweep [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_vol.plot_cappi(radar, field_name, altitude, prdcfg, fname_list, save_fig=True)`
plots a Constant Altitude Plan Position Indicator CAPPI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

altitude [float] the altitude [m MSL] to be plotted

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tuple] list of names of the saved plots or handle of the figure and axes

`pyrad.graph.plots_vol.plot_field_coverage` (*xval_list*, *yval_list*, *fname_list*, *labelx*='Azimuth (deg)', *labely*='Range extension [m]', *labels*=None, *title*='Field coverage', *ymin*=None, *ymax*=None, *xmeanval*=None, *ymeanval*=None, *labelmeanval*=None, *dpi*=72)

plots a time series

Parameters

xval_list [list of float arrays] the x values, azimuth

yval_list [list of float arrays] the y values. Range extension

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

labels [array of str] The label of the legend

title [str] The figure title

ymin, ymax [float] Lower/Upper limit of y axis

xmeanval, ymeanval [float array] the x and y values of a mean along elevation

labelmeanval [str] the label of the mean

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_vol.plot_pos` (*lat*, *lon*, *alt*, *fname_list*, *ax*=None, *fig*=None, *save_fig*=True, *sort_altitude*='No', *dpi*=72, *alpha*=1.0, *cb_label*='height [m MSL]', *titl*='Position', *xlabel*='Lon [Deg]', *ylabel*='Lat [Deg]', *limits*=None, *vmin*=None, *vmax*=None)

plots a trajectory on a Cartesian surface

Parameters

lat, lon, alt [float array] Points coordinates

fname_list [list of str] list of names of the files where to store the plot

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

sort_altitude [str] String indicating whether to sort the altitude data. Can be 'No', 'Lowest_on_top' or 'Highest_on_top'

dpi [int] Pixel density

alpha [float] Transparency

cb_label [str] Color bar label

titl [str] Plot title

limits [tuple or None] The limits of the field to plot

Returns

fname_list [list of str or]

fig, ax [tuple] list of names of the saved plots or handle of the figure and axes

`pyrad.graph.plots_vol.plot_ppi(radar, field_name, ind_el, prdcfg, fname_list, plot_type='PPI', titl=None, step=None, quantiles=None, save_fig=True)`
plots a PPI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_el [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

plot_type [str] type of plot (PPI, QUANTILES or HISTOGRAM)

titl [str] Plot title

step [float] step for histogram plotting

quantiles [float array] quantiles to plot

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_vol.plot_ppi_contour(radar, field_name, ind_el, prdcfg, fname_list, contour_values=None, linewidths=1.5, ax=None, fig=None, save_fig=True)`

plots contour data on a PPI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_el [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

contour_values [float array] list of contours to plot

linewidths [float] width of the contour lines

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tuple] list of names of the saved plots or handle of the figure an axes

`pyrad.graph.plots_vol.plot_ppi_map(radar, field_name, ind_el, prdcfg, fname_list)`
plots a PPI on a geographic map

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_el [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_vol.plot_rhi(radar, field_name, ind_az, prdcfg, fname_list, plot_type='RHI',
 titl=None, step=None, quantiles=None, save_fig=True)`
plots an RHI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_az [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

plot_type [str] type of plot (PPI, QUANTILES or HISTOGRAM)

titl [str] Plot title

step [float] step for histogram plotting

quantiles [float array] quantiles to plot

save_fig [bool] if true save the figure. If false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str] list of names of the created plots

fig, ax [tuple] list of names of the saved plots or handle of the figure an axes

`pyrad.graph.plots_vol.plot_rhi_contour(radar, field_name, ind_az, prdcfg, fname_list,
 contour_values=None, linewidths=1.5, ax=None,
 fig=None, save_fig=True)`
plots contour data on an RHI

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_az [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

contour_values [float array] list of contours to plot

linewidths [float] width of the contour lines

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tuple] list of names of the saved plots or handle of the figure an axes

`pyrad.graph.plots_vol.plot_rhi_profile`(*data_list, hvec, fname_list, labelx='Value', labely='Height (m MSL)', labels=['Mean'], title='RHI profile', colors=None, linestyle=None, vmin=None, vmax=None, hmin=None, hmax=None, dpi=72*)

plots an RHI profile

Parameters

data_list [list of float array] values of the profile

hvec [float array] height points of the profile

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

labels [array of str] The label of the legend

title [str] The figure title

colors [array of str] Specifies the colors of each line

linestyle [array of str] Specifies the line style of each line

vmin, vmax: float Lower/Upper limit of data values

hmin, hmax: float Lower/Upper limit of altitude

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_vol.plot_time_range`(*radar, field_name, ind_sweep, prdcfg, fname_list*)

plots a time-range plot

Parameters

radar [Radar object] object containing the radar data to plot

field_name [str] name of the radar field to plot

ind_sweep [int] sweep index to plot

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_vol.plot_traj(rng_traj, azi_traj, ele_traj, time_traj, prdcfg, fname_list, rad_alt=None, rad_tstart=None, ax=None, fig=None, save_fig=True)`

plots a trajectory on a Cartesian surface

Parameters

rng_traj, azi_traj, ele_traj [float array] antenna coordinates of the trajectory [m and deg]

time_traj [datetime array] trajectory time

prdcfg [dict] dictionary containing the product configuration

fname_list [list of str] list of names of the files where to store the plot

rad_alt [float or None] radar altitude [m MSL]

rad_tstart [datetime object or None] start time of the radar scan

surface_alt [float] surface altitude [m MSL]

color_ref [str] What the color code represents. Can be 'None', 'rel_altitude', 'altitude' or 'time'

fig [Figure] Figure to add the colorbar to. If none a new figure will be created

ax [Axis] Axis to plot on. if fig is None a new axis will be created

save_fig [bool] if true save the figure if false it does not close the plot and returns the handle to the figure

Returns

fname_list [list of str or]

fig, ax [tuple] list of names of the saved plots or handle of the figure an axes

PYRAD.GRAPH.PLOTS_GRID

Functions to plot data in a Cartesian grid format

<code>plot_surface(grid, field_name, level, ...)</code>	plots a surface from gridded data
<code>plot_latitude_slice(grid, field_name, lon, ...)</code>	plots a latitude slice from gridded data
<code>plot_longitude_slice(grid, field_name, lon, ...)</code>	plots a longitude slice from gridded data
<code>plot_latlon_slice(grid, field_name, coord1, ...)</code>	plots a croos section crossing two points in the grid

`pyrad.graph.plots_grid.plot_latitude_slice(grid, field_name, lon, lat, prdcfg, fname_list)`
plots a latitude slice from gridded data

Parameters

- grid** [Grid object] object containing the gridded data to plot
- field_name** [str] name of the radar field to plot
- lon, lat** [float] coordinates of the slice to plot
- prdcfg** [dict] dictionary containing the product configuration
- fname_list** [list of str] list of names of the files where to store the plot

Returns

- fname_list** [list of str] list of names of the created plots

`pyrad.graph.plots_grid.plot_latlon_slice(grid, field_name, coord1, coord2, prdcfg, fname_list)`
plots a croos section crossing two points in the grid

Parameters

- grid** [Grid object] object containing the gridded data to plot
- field_name** [str] name of the radar field to plot
- coord1** [tuple of floats] lat, lon of the first point
- coord2** [tuple of floats] lat, lon of the second point
- fname_list** [list of str] list of names of the files where to store the plot

Returns

- fname_list** [list of str] list of names of the created plots

`pyrad.graph.plots_grid.plot_longitude_slice(grid, field_name, lon, lat, prdcfg, fname_list)`
plots a longitude slice from gridded data

Parameters

grid [Grid object] object containing the gridded data to plot
field_name [str] name of the radar field to plot
lon, lat [float] coordinates of the slice to plot
prdcfg [dict] dictionary containing the product configuration
fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_grid.plot_surface(grid, field_name, level, prdcfg, fname_list)`
plots a surface from gridded data

Parameters

grid [Grid object] object containing the gridded data to plot
field_name [str] name of the radar field to plot
level [int] level index
prdcfg [dict] dictionary containing the product configuration
fname_list [list of str] list of names of the files where to store the plot

Returns

fname_list [list of str] list of names of the created plots

PYRAD.GRAPH.PLOT_TIMESERIES

Functions to plot Pyrad datasets

<code>plot_timeseries(tvec, data_list, fname_list)</code>	plots a time series
<code>plot_timeseries_comp(date1, value1, date2, ...)</code>	plots 2 time series in the same graph
<code>plot_monitoring_ts(date, np_t, cquant, ...)</code>	plots a time series of monitoring data
<code>plot_intercomp_scores_ts(date_vec, np_vec, ...)</code>	plots a time series of radar intercomparison scores
<code>plot_ml_ts(dt_ml_arr, ml_top_avg_arr, ...[, ...])</code>	plots a time series of melting layer data
<code>plot_sun_retrieval_ts(sun_retrieval, ...[, ...])</code>	plots sun retrieval time series series

```
pyrad.graph.plots_timeseries.plot_intercomp_scores_ts(date_vec, np_vec, mean-
                                                         bias_vec, median-
                                                         bias_vec, quant25bias_vec,
                                                         quant75bias_vec, mode-
                                                         bias_vec, corr_vec,
                                                         slope_vec, intercep_vec,
                                                         intercep_slope1_vec,
                                                         fname_list, ref_value=0.0,
                                                         np_min=0, corr_min=0.0,
                                                         labelx='Time UTC',
                                                         titl='RADAR001-RADAR002
                                                         intercomparison', dpi=72)
```

plots a time series of radar intercomparison scores

Parameters

date_vec [datetime object] time of the time series

np_vec [int array] number of points

meanbias_vec, medianbias_vec, modebias_vec [float array] mean, median and mode bias

quant25bias_vec, quant75bias_vec: 25th and 75th percentile of the bias

corr_vec [float array] correlation

slope_vec, intercep_vec [float array] slope and intercep of a linear regression

intercep_slope1_vec [float] the intercep point of a inear regression of slope 1

ref_value [float] the reference value

np_min [int] The minimum number of points to consider the result valid

corr_min [float] The minimum correlation to consider the results valid

labelx [str] The label of the X axis

titl [str] The figure title

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_timeseries.plot_ml_ts` (*dt_ml_arr, ml_top_avg_arr, ml_top_std_arr, thick_avg_arr, thick_std_arr, nrays_valid_arr, nrays_total_arr, fname_list, labelx='Time UTC', titl='Melting layer time series', dpi=72*)

plots a time series of melting layer data

Parameters

dt_ml_arr [datetime object] time of the time series

np_vec [int array] number of points

meanbias_vec, medianbias_vec, modebias_vec [float array] mean, median and mode bias

quant25bias_vec, quant75bias_vec: 25th and 75th percentile of the bias

corr_vec [float array] correlation

slope_vec, intercep_vec [float array] slope and intercep of a linear regression

intercep_slope1_vec [float] the intercep point of a inear regression of slope 1

ref_value [float] the reference value

np_min [int] The minimum number of points to consider the result valid

corr_min [float] The minimum correlation to consider the results valid

labelx [str] The label of the X axis

titl [str] The figure title

Returns

fname_list [list of str] list of names of the created plots

`pyrad.graph.plots_timeseries.plot_monitoring_ts` (*date, np_t, cquant, lquant, hquant, field_name, fname_list, ref_value=None, vmin=None, vmax=None, np_min=0, labelx='Time [UTC]', labely='Value', titl='Time Series', dpi=72*)

plots a time series of monitoring data

Parameters

date [datetime object] time of the time series

np_t [int array] number of points

cquant, lquant, hquant [float array] values of the central, low and high quantiles

field_name [str] name of the field

fname_list [list of str] list of names of the files where to store the plot

ref_value [float] the reference value

vmin, vmax [float] The limits of the y axis

np_min [int] minimum number of points to consider the sample plotable

labelx [str] The label of the X axis

labeledy [str] The label of the Y axis

titl [str] The figure title

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

```
pyrad.graph.plots_timeseries.plot_sun_retrieval_ts(sun_retrieval, data_type,  
                                                    fname_list, labelx='Date',  
                                                    titl='Sun retrieval Time Series',  
                                                    dpi=72)
```

plots sun retrieval time series series

Parameters

sun_retrieval [tuple] tuple containing the retrieved parameters

data_type [str] parameter to be plotted

fname_list [list of str] list of names of the files where to store the plot

labelx [str] the x label

titl [str] the title of the plot

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

```
pyrad.graph.plots_timeseries.plot_timeseries(tvec, data_list, fname_list, labelx='Time  
[UTC]', labeledy='Value', labels=['Sensor'],  
                                              title='Time Series', period=0, timeformat  
                                              mat=None, colors=None, linestyles=None,  
                                              markers=None, ymin=None, ymax=None,  
                                              dpi=72)
```

plots a time series

Parameters

tvec [datetime object] time of the time series

data_list [list of float array] values of the time series

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labeledy [str] The label of the Y axis

labels [array of str] The label of the legend

title [str] The figure title

period [float] measurement period in seconds used to compute accumulation. If 0 no accumulation is computed

timeformat [str] Specifies the tvec and time format on the x axis

colors [array of str] Specifies the colors of each line

linestyles [array of str] Specifies the line style of each line

markers: array of str Specify the markers to be used for each line

ymin, ymax: float Lower/Upper limit of y axis

dpi [int] dots per inch

Returns

fname_list [list of str] list of names of the created plots

```
pyrad.graph.plots_timeseries.plot_timeseries_comp(date1, value1, date2, value2,  
                                                  fname_list, labelx='Time [UTC]',  
                                                  labely='Value', label1='Sensor  
1', label2='Sensor 2', titl='Time  
Series Comparison', period1=0, pe-  
riod2=0, ymin=None, ymax=None,  
                                                  dpi=72)
```

plots 2 time series in the same graph

Parameters

date1 [datetime object] time of the first time series

value1 [float array] values of the first time series

date2 [datetime object] time of the second time series

value2 [float array] values of the second time series

fname_list [list of str] list of names of the files where to store the plot

labelx [str] The label of the X axis

labely [str] The label of the Y axis

label1, label2 [str] legend label for each time series

titl [str]

The figure title

period1, period2 [float] measurement period in seconds used to compute accumulation. If
0 no accumulation is computed

dpi [int] dots per inch

ymin, ymax [float] The limits of the Y-axis. None will keep the default limit.

Returns

fname_list [list of str] list of names of the created plots

PYRAD.UTIL.RADAR_UTILS

Miscellaneous functions dealing with radar data

<i>get_ROI</i> (radar, fieldname, sector)	filter out any data outside the region of interest defined by sector
<i>rainfall_accumulation</i> (t_in_vec, val_in_vec)	Computes the rainfall accumulation of a time series over a given period
<i>time_series_statistics</i> (t_in_vec, val_in_vec)	Computes statistics over a time-averaged series
<i>join_time_series</i> (t1, val1, t2, val2[, dropnan])	joins time_series
<i>get_range_bins_to_avg</i> (rad1_rng, rad2_rng)	Compares the resolution of two radars and determines if and which radar has to be averaged and the length of the averaging window
<i>belongs_roi_indices</i> (lat, lon, roi)	Get the indices of points that belong to roi in a list of points
<i>find_ray_index</i> (ele_vec, azi_vec, ele, azi[, ...])	Find the ray index corresponding to a particular elevation and azimuth
<i>find_rng_index</i> (rng_vec, rng[, rng_tol])	Find the range index corresponding to a particular range
<i>find_nearest_gate</i> (radar, lat, lon[, latlon_tol])	Find the radar gate closest to a lat,lon point
<i>find_neighbour_gates</i> (radar, azi, rng[, ...])	Find the neighbouring gates within +-delta_azi and +-delta_rng
<i>find_colocated_indexes</i> (radar1, radar2, ...)	Given the theoretical elevation, azimuth and range of the co-located gates of two radars and a given tolerance returns the indices of the gates for the current radars
<i>get_target_elevations</i> (radar_in)	Gets RHI target elevations
<i>time_avg_range</i> (timeinfo, avg_starttime, ...)	finds the new start and end time of an averaging
<i>get_closest_solar_flux</i> (hit_datetime_list, ...)	finds the solar flux measurement closest to the sun hit
<i>create_sun_hits_field</i> (rad_el, rad_az, ...)	creates a sun hits field from the position and power of the sun hits
<i>create_sun_retrieval_field</i> (par, field_name, ...)	creates a sun retrieval field from the retrieval parameters
<i>compute_quantiles</i> (field[, quantiles])	computes quantiles
<i>compute_quantiles_from_hist</i> (bin_centers, hist)	computes quantiles from histograms
<i>compute_quantiles_sweep</i> (field, ray_start, ...)	computes quantiles of a particular sweep
<i>compute_histogram</i> (field, field_name[, ...])	computes histogram of the data
<i>compute_histogram_sweep</i> (field, ray_start, ...)	computes histogram of the data in a particular sweep
<i>get_histogram_bins</i> (field_name[, step])	gets the histogram bins using the range limits of the field as defined in the Py-ART config file.
<i>compute_2d_stats</i> (field1, field2, ...[, ...])	computes a 2D histogram and statistics of the data

Continued on next page

Table 1 – continued from previous page

<code>compute_1d_stats</code> (field1, field2)	returns statistics of data
<code>compute_2d_hist</code> (field1, field2, field_name1, ...)	computes a 2D histogram of the data
<code>quantize_field</code> (field, field_name, step)	quantizes data
<code>compute_profile_stats</code> (field, gate_altitude, ...)	Compute statistics of vertical profile
<code>compute_directional_stats</code> (field[, avg_type, ...])	Computes the mean or the median along one of the axis (ray or range)
<code>project_to_vertical</code> (data_in, data_height, ...)	Projects radar data to a regular vertical grid

`pyrad.util.radar_utils.belongs_roi_indices` (*lat, lon, roi*)

Get the indices of points that belong to roi in a list of points

Parameters

lat, lon [float arrays] latitudes and longitudes to check

roi [dict] Dictionary describing the region of interest

Returns

inds [array of ints] list of indices of points belonging to ROI

is_roi [str] Whether the list of points is within the region of interest. Can be 'All', 'None', 'Some'

`pyrad.util.radar_utils.compute_1d_stats` (*field1, field2*)

returns statistics of data

Parameters

field1, field2 [ndarray 1D] the two fields to compare

Returns

stats [dict] a dictionary with statistics

`pyrad.util.radar_utils.compute_2d_hist` (*field1, field2, field_name1, field_name2, step1=None, step2=None*)

computes a 2D histogram of the data

Parameters

field1, field2 [ndarray 2D] the radar fields

field_name1, field_name2 [str] field names

step1, step2 [float] size of the bins

Returns

H [float array 2D] The bi-dimensional histogram of samples x and y

xedges, yedges [float array] the bin edges along each dimension

`pyrad.util.radar_utils.compute_2d_stats` (*field1, field2, field_name1, field_name2, step1=None, step2=None*)

computes a 2D histogram and statistics of the data

Parameters

field1, field2 [ndarray 2D] the two fields

field_name1, field_name2: str the name of the fields

step1, step2 [float] size of bin

Returns

hist_2d [array] the histogram

bin_edges1, bin_edges2 [float array] The bin edges

stats [dict] a dictionary with statistics

`pyrad.util.radar_utils.compute_directional_stats` (*field*, *avg_type='mean'*,
nvalid_min=1, *axis=0*)

Computes the mean or the median along one of the axis (ray or range)

Parameters

field [ndarray] the radar field

avg_type: str the type of average: 'mean' or 'median'

nvalid_min [int] the minimum number of points to consider the stats valid. Default 1

axis [int] the axis along which to compute (0=ray, 1=range)

Returns

values [ndarray 1D] The resultant statistics

nvalid [ndarray 1D] The number of valid points used in the computation

`pyrad.util.radar_utils.compute_histogram` (*field*, *field_name*, *bin_edges=None*, *step=None*,
vmin=None, *vmax=None*)

computes histogram of the data

Parameters

field [ndarray 2D] the radar field

field_name: str or none name of the field

bins_edges: ndarray 1D the bin edges

step [float] size of bin

vmin, vmax [float] The minimum and maximum value of the histogram

Returns

bin_edges [float array] interval of each bin

values [float array] values at each bin

`pyrad.util.radar_utils.compute_histogram_sweep` (*field*, *ray_start*, *ray_end*, *field_name*,
step=None)

computes histogram of the data in a particular sweep

Parameters

field [ndarray 2D] the radar field

ray_start, ray_end [int] starting and ending ray indexes

field_name: str name of the field

step [float] size of bin

Returns

bin_edges [float array] interval of each bin

values [float array] values at each bin

`pyrad.util.radar_utils.compute_profile_stats` (*field, gate_altitude, h_vec, h_res, quantity='quantiles', quantiles=array([0.25, 0.5, 0.75]), nvalid_min=4, std_field=None, np_field=None, make_linear=False, include_nans=False*)

Compute statistics of vertical profile

Parameters

field [ndarray] the radar field

gate_altitude: ndarray the altitude at each radar gate [m MSL]

h_vec [1D ndarray] height vector [m MSL]

h_res [float] height resolution [m]

quantity [str] The quantity to compute. Can be ['quantiles', 'mode', 'regression_mean', 'mean']. If 'mean', the min, max, and average is computed.

quantiles [1D ndarray] the quantiles to compute

nvalid_min [int] the minimum number of points to consider the stats valid

std_field [ndarray] the standard deviation of the regression at each range gate

np_field [ndarray] the number of points used to compute the regression at each range gate

make_linear [Boolean] If true the data is transformed into linear coordinates before taking the mean

include_nans [Boolean] If true NaN will be considered as zeros

Returns

vals [ndarray 2D] The resultant statistics

val_valid [ndarray 1D] The number of points to compute the stats used at each height level

`pyrad.util.radar_utils.compute_quantiles` (*field, quantiles=None*)
computes quantiles

Parameters

field [ndarray 2D] the radar field

ray_start, ray_end [int] starting and ending ray indexes

quantiles: float array list of quantiles to compute

Returns

quantiles [float array] list of quantiles

values [float array] values at each quantile

`pyrad.util.radar_utils.compute_quantiles_from_hist` (*bin_centers, hist, quantiles=None*)
computes quantiles from histograms

Parameters

bin_centers [ndarray 1D] the bins

hist [ndarray 1D] the histogram

quantiles: float array list of quantiles to compute

Returns

quantiles [float array] list of quantiles

values [float array] values at each quantile

`pyrad.util.radar_utils.compute_quantiles_sweep` (*field, ray_start, ray_end, quantiles=None*)

computes quantiles of a particular sweep

Parameters

field [ndarray 2D] the radar field

ray_start, ray_end [int] starting and ending ray indexes

quantiles: float array list of quantiles to compute

Returns

quantiles [float array] list of quantiles

values [float array] values at each quantile

`pyrad.util.radar_utils.create_sun_hits_field` (*rad_el, rad_az, sun_el, sun_az, data, imgcfg*)

creates a sun hits field from the position and power of the sun hits

Parameters

rad_el, rad_az, sun_el, sun_az [ndarray 1D] azimuth and elevation of the radar and the sun respectively in degree

data [masked ndarray 1D] the sun hit data

imgcfg: dict a dictionary specifying the ranges and resolution of the field to create

Returns

field [masked ndarray 2D] the sun hit field

`pyrad.util.radar_utils.create_sun_retrieval_field` (*par, field_name, imgcfg, lant=0.0*)

creates a sun retrieval field from the retrieval parameters

Parameters

par [ndarray 1D] the 5 retrieval parameters

imgcfg: dict a dictionary specifying the ranges and resolution of the field to create

Returns

field [masked ndarray 2D] the sun retrieval field

`pyrad.util.radar_utils.find_colocated_indexes` (*radar1, radar2, rad1_ele, rad1_azi, rad1_rng, rad2_ele, rad2_azi, rad2_rng, ele_tol=0.5, azi_tol=0.5, rng_tol=50.0*)

Given the theoretical elevation, azimuth and range of the co-located gates of two radars and a given tolerance returns the indices of the gates for the current radars

Parameters

radar1, radar2 [radar objects] the two radar objects

rad1_ele, rad1_azi, rad1_rng [array of floats] the radar coordinates of the radar1 gates

rad2_ele, rad2_azi, rad2_rng [array of floats] the radar coordinates of the radar2 gates

ele_tol, azi_tol [floats] azimuth and elevation angle tolerance [deg]

rng_tol [float] range Tolerance [m]

Returns

ind_ray_rad1, ind_rng_rad1, ind_ray_rad2, ind_rng_rad2 [array of ints] the ray and range indexes of each radar gate

`pyrad.util.radar_utils.find_nearest_gate(radar, lat, lon, latlon_tol=0.0005)`

Find the radar gate closest to a lat,lon point

Parameters

radar [radar object] the radar object

lat, lon [float] The position of the point

latlon_tol [float] The tolerance around this point

Returns

ind_ray, ind_rng [int] The ray and range index

azi, rng [float] the range and azimuth position of the gate

`pyrad.util.radar_utils.find_neighbour_gates(radar, azi, rng, delta_azi=None, delta_rng=None)`

Find the neighbouring gates within +-delta_azi and +-delta_rng

Parameters

radar [radar object] the radar object

azi, rng [float] The azimuth [deg] and range [m] of the central gate

delta_azi, delta_rng [float] The extend where to look for

Returns

inds_ray_aux, ind_rng_aux [int] The indices (ray, rng) of the neighbouring gates

`pyrad.util.radar_utils.find_ray_index(ele_vec, azi_vec, ele, azi, ele_tol=0.0, azi_tol=0.0, nearest='azi')`

Find the ray index corresponding to a particular elevation and azimuth

Parameters

ele_vec, azi_vec [float arrays] The elevation and azimuth data arrays where to look for

ele, azi [floats] The elevation and azimuth to search

ele_tol, azi_tol [floats] Tolerances [deg]

nearest [str] criteria to define wich ray to keep if multiple rays are within tolerance. azi: nearest azimuth, ele: nearest elevation

Returns

ind_ray [int] The ray index

`pyrad.util.radar_utils.find_rng_index(rng_vec, rng, rng_tol=0.0)`

Find the range index corresponding to a particular range

Parameters

rng_vec [float array] The range data array where to look for

rng [float] The range to search

rng_tol [float] Tolerance [m]

Returns

ind_rng [int] The range index

`pyrad.util.radar_utils.get_ROI (radar, fieldname, sector)`
filter out any data outside the region of interest defined by sector

Parameters

radar [radar object] the radar object where the data is

fieldname [str] name of the field to filter

sector [dict] a dictionary defining the region of interest

Returns

roi_flag [ndarray] a field array with ones in gates that are in the Region of Interest

`pyrad.util.radar_utils.get_closest_solar_flux (hit_datetime_list, flux_datetime_list, flux_value_list)`

finds the solar flux measurement closest to the sun hit

Parameters

hit_datetime_list [datetime array] the date and time of the sun hit

flux_datetime_list [datetime array] the date and time of the solar flux measurement

flux_value_list: ndarray 1D the solar flux values

Returns

flux_datetime_closest_list [datetime array] the date and time of the solar flux measurement closest to sun hit

flux_value_closest_list [ndarray 1D] the solar flux values closest to the sun hit time

`pyrad.util.radar_utils.get_histogram_bins (field_name, step=None)`
gets the histogram bins using the range limits of the field as defined in the Py-ART config file.

Parameters

field_name: str name of the field

step [float] size of bin

Returns

bin_edges [float array] The bin edges

`pyrad.util.radar_utils.get_range_bins_to_avg (rad1_rng, rad2_rng)`

Compares the resolution of two radars and determines if and which radar has to be averaged and the length of the averaging window

Parameters

rad1_rng [array] the range of radar 1

rad2_rng [datetime] the range of radar 2

Returns

avg_rad1, avg_rad2 [Boolean] Booleans specifying if the radar data has to be average in range

avg_rad_lim [array with two elements] the limits to the average (centered on each range gate)

`pyrad.util.radar_utils.get_target_elevations (radar_in)`

Gets RHI target elevations

Parameters

radar_in [Radar object] current radar object

Returns

target_elevations [1D-array] Azimuth angles

el_tol [float] azimuth tolerance

`pyrad.util.radar_utils.join_time_series(t1, val1, t2, val2, dropnan=False)`
joins time_series

Parameters

t1 [datetime array] time of first series

val1 [float array] value of first series

t2 [datetime array] time of second series

val2 [float array] value of second series

dropnan [boolean] if True remove NaN from the time series

Returns

t_out_vec [datetime array] the resultant date time after joining the series

val1_out_vec [float array] value of first series

val2_out_vec [float array] value of second series

`pyrad.util.radar_utils.project_to_vertical(data_in, data_height, grid_height, interp_kind='none', fill_value=-9999.0)`

Projects radar data to a regular vertical grid

Parameters

data_in [ndarray 1D] the radar data to project

data_height [ndarray 1D] the height of each radar point

grid_height [ndarray 1D] the regular vertical grid to project to

interp_kind [str] The type of interpolation to use: 'none' or 'nearest'

fill_value [float] The fill value used for interpolation

Returns

data_out [ndarray 1D] The projected data

`pyrad.util.radar_utils.quantize_field(field, field_name, step)`
quantizes data

Parameters

field [ndarray 2D] the radar field

field_name: str name of the field

step [float] size of bin

Returns

fieldq [ndarray 2D] The quantized field

values [float array] values at each bin

`pyrad.util.radar_utils.rainfall_accumulation(t_in_vec, val_in_vec, cum_time=3600.0, base_time=0.0, dropnan=False)`

Computes the rainfall accumulation of a time series over a given period

Parameters

t_in_vec [datetime array] the input date and time array
val_in_vec [float array] the input values array [mm/h]
cum_time [int] accumulation time [s]
base_time [int] base time [s]
dropnan [boolean] if True remove NaN from the time series

Returns

t_out_vec [datetime array] the output date and time array
val_out_vec [float array] the output values array
np_vec [int array] the number of samples at each period

`pyrad.util.radar_utils.time_avg_range(timeinfo, avg_starttime, avg_endtime, period)`
finds the new start and end time of an averaging

Parameters

timeinfo [datetime] the current volume time
avg_starttime [datetime] the current average start time
avg_endtime: datetime the current average end time
period: float the averaging period

Returns

new_starttime [datetime] the new average start time
new_endtime [datetime] the new average end time

`pyrad.util.radar_utils.time_series_statistics(t_in_vec, val_in_vec, avg_time=3600, base_time=1800, method='mean', dropnan=False)`

Computes statistics over a time-averaged series

Parameters

t_in_vec [datetime array] the input date and time array
val_in_vec [float array] the input values array
avg_time [int] averaging time [s]
base_time [int] base time [s]
method [str] statistical method
dropnan [boolean] if True remove NaN from the time series

Returns

t_out_vec [datetime array] the output date and time array
val_out_vec [float array] the output values array

PYRAD.UTIL.STAT_UTILS

Miscellaneous functions dealing with statistics

<i>quantiles_weighted</i> (values[, ...])	weight_vector,	Given a set of values and weights, compute the weighted quantile(s).
--	----------------	---

```
pyrad.util.stat_utils.quantiles_weighted(values,      weight_vector=None,      quan-  
                                           tiles=array([0.5]),      weight_threshold=None,  
                                           data_is_log=False)  
    Given a set of values and weights, compute the weighted quantile(s).
```


INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

- pyrad.flow.flow_aux, 1
- pyrad.flow.flow_control, 7
- pyrad.graph.plots, 159
- pyrad.graph.plots_aux, 158
- pyrad.graph.plots_grid, 171
- pyrad.graph.plots_timeseries, 174
- pyrad.graph.plots_vol, 164
- pyrad.io.config, 90
- pyrad.io.io_aux, 83
- pyrad.io.mxpole_config, 120
- pyrad.io.read_data_cosmo, 120
- pyrad.io.read_data_hzt, 123
- pyrad.io.read_data_mxpole, 99
- pyrad.io.read_data_other, 134
- pyrad.io.read_data_radar, 92
- pyrad.io.read_data_sensor, 126
- pyrad.io.read_data_sun, 131
- pyrad.io.timeseries, 148
- pyrad.io.trajectory, 152
- pyrad.io.write_data, 140
- pyrad.proc.process_aux, 10
- pyrad.proc.process_calib, 13
- pyrad.proc.process_cosmo, 18
- pyrad.proc.process_Doppler, 21
- pyrad.proc.process_echoclass, 25
- pyrad.proc.process_intercomp, 31
- pyrad.proc.process_monitoring, 36
- pyrad.proc.process_phase, 41
- pyrad.proc.process_retrieve, 47
- pyrad.proc.process_timeseries, 53
- pyrad.proc.process_traj, 59
- pyrad.prod.process_grid_products, 73
- pyrad.prod.process_intercomp_products, 79
- pyrad.prod.process_monitoring_products, 77
- pyrad.prod.process_product, 69
- pyrad.prod.process_timeseries_products, 75
- pyrad.prod.process_traj_products, 82
- pyrad.prod.process_vol_products, 72
- pyrad.prod.product_aux, 67
- pyrad.util.radar_utils, 178
- pyrad.util.stat_utils, 187

Symbols

[_DataSeries \(class in pyrad.io.timeseries\)](#), 151
[_Radar_Trajectory \(class in pyrad.io.trajectory\)](#), 156
[__class__ \(pyrad.io.read_data_mxpole.pyrad_IDL attribute\)](#), 103
[__class__ \(pyrad.io.read_data_mxpole.pyrad_MCH attribute\)](#), 109
[__class__ \(pyrad.io.read_data_mxpole.pyrad_MXPOL attribute\)](#), 114
[__class__ \(pyrad.io.timeseries.TimeSeries attribute\)](#), 149
[__class__ \(pyrad.io.timeseries._DataSeries attribute\)](#), 151
[__class__ \(pyrad.io.trajectory.Trajectory attribute\)](#), 154
[__class__ \(pyrad.io.trajectory._Radar_Trajectory attribute\)](#), 156
[__class__ \(pyrad.proc.process_traj.TargetRadar attribute\)](#), 61
[__delattr__ \(pyrad.io.read_data_mxpole.pyrad_IDL attribute\)](#), 103
[__delattr__ \(pyrad.io.read_data_mxpole.pyrad_MCH attribute\)](#), 109
[__delattr__ \(pyrad.io.read_data_mxpole.pyrad_MXPOL attribute\)](#), 114
[__delattr__ \(pyrad.io.timeseries.TimeSeries attribute\)](#), 149
[__delattr__ \(pyrad.io.timeseries._DataSeries attribute\)](#), 151
[__delattr__ \(pyrad.io.trajectory.Trajectory attribute\)](#), 154
[__delattr__ \(pyrad.io.trajectory._Radar_Trajectory attribute\)](#), 156
[__delattr__ \(pyrad.proc.process_traj.TargetRadar attribute\)](#), 61
[__dict__ \(pyrad.io.read_data_mxpole.pyrad_IDL attribute\)](#), 103
[__dict__ \(pyrad.io.read_data_mxpole.pyrad_MCH attribute\)](#), 109
[__dict__ \(pyrad.io.read_data_mxpole.pyrad_MXPOL attribute\)](#), 114
[__dict__ \(pyrad.io.timeseries.TimeSeries attribute\)](#), 149
[__dict__ \(pyrad.io.timeseries._DataSeries attribute\)](#), 151
[__dict__ \(pyrad.io.trajectory.Trajectory attribute\)](#), 154
[__dict__ \(pyrad.io.trajectory._Radar_Trajectory attribute\)](#), 156
[__dict__ \(pyrad.proc.process_traj.TargetRadar attribute\)](#), 61
[__dir__ \(\) \(pyrad.io.read_data_mxpole.pyrad_IDL method\)](#), 103
[__dir__ \(\) \(pyrad.io.read_data_mxpole.pyrad_MCH method\)](#), 109
[__dir__ \(\) \(pyrad.io.read_data_mxpole.pyrad_MXPOL method\)](#), 114
[__dir__ \(\) \(pyrad.io.timeseries.TimeSeries method\)](#), 149
[__dir__ \(\) \(pyrad.io.timeseries._DataSeries method\)](#), 151
[__dir__ \(\) \(pyrad.io.trajectory.Trajectory method\)](#), 154
[__dir__ \(\) \(pyrad.io.trajectory._Radar_Trajectory method\)](#), 156
[__dir__ \(\) \(pyrad.proc.process_traj.TargetRadar method\)](#), 61
[__eq__ \(pyrad.io.read_data_mxpole.pyrad_IDL attribute\)](#), 103
[__eq__ \(pyrad.io.read_data_mxpole.pyrad_MCH attribute\)](#), 109
[__eq__ \(pyrad.io.read_data_mxpole.pyrad_MXPOL attribute\)](#), 114
[__eq__ \(pyrad.io.timeseries.TimeSeries attribute\)](#), 149
[__eq__ \(pyrad.io.timeseries._DataSeries attribute\)](#), 151
[__eq__ \(pyrad.io.trajectory.Trajectory attribute\)](#), 154
[__eq__ \(pyrad.io.trajectory._Radar_Trajectory attribute\)](#), 156
[__eq__ \(pyrad.proc.process_traj.TargetRadar attribute\)](#), 61
[__format__ \(\) \(pyrad.io.read_data_mxpole.pyrad_IDL method\)](#), 103
[__format__ \(\) \(pyrad.io.read_data_mxpole.pyrad_MCH method\)](#), 109

`method`), 109
`__format__()` (`pyrad.io.read_data_mxpole.pyrad_MXPOL` `method`), 114
`__format__()` (`pyrad.io.timeseries.TimeSeries` `method`), 149
`__format__()` (`pyrad.io.timeseries._DataSeries` `method`), 151
`__format__()` (`pyrad.io.trajectory.Trajectory` `method`), 154
`__format__()` (`pyrad.io.trajectory._Radar_Trajectory` `method`), 156
`__format__()` (`pyrad.proc.process_traj.TargetRadar` `method`), 61
`__ge__` (`pyrad.io.read_data_mxpole.pyrad_IDL` `attribute`), 104
`__ge__` (`pyrad.io.read_data_mxpole.pyrad_MCH` `attribute`), 109
`__ge__` (`pyrad.io.read_data_mxpole.pyrad_MXPOL` `attribute`), 114
`__ge__` (`pyrad.io.timeseries.TimeSeries` `attribute`), 149
`__ge__` (`pyrad.io.timeseries._DataSeries` `attribute`), 151
`__ge__` (`pyrad.io.trajectory.Trajectory` `attribute`), 154
`__ge__` (`pyrad.io.trajectory._Radar_Trajectory` `attribute`), 157
`__ge__` (`pyrad.proc.process_traj.TargetRadar` `attribute`), 62
`__getattr__` (`pyrad.io.read_data_mxpole.pyrad_IDL` `attribute`), 104
`__getattr__` (`pyrad.io.read_data_mxpole.pyrad_MCH` `attribute`), 109
`__getattr__` (`pyrad.io.read_data_mxpole.pyrad_MXPOL` `attribute`), 114
`__getattr__` (`pyrad.io.timeseries.TimeSeries` `attribute`), 150
`__getattr__` (`pyrad.io.timeseries._DataSeries` `attribute`), 151
`__getattr__` (`pyrad.io.trajectory.Trajectory` `attribute`), 154
`__getattr__` (`pyrad.io.trajectory._Radar_Trajectory` `attribute`), 157
`__getattr__` (`pyrad.proc.process_traj.TargetRadar` `attribute`), 62
`__getstate__()` (`pyrad.io.read_data_mxpole.pyrad_IDL` `method`), 104
`__getstate__()` (`pyrad.io.read_data_mxpole.pyrad_MCH` `method`), 109
`__getstate__()` (`pyrad.io.read_data_mxpole.pyrad_MXPOL` `method`), 114
`__gt__` (`pyrad.io.read_data_mxpole.pyrad_IDL` `attribute`), 104
`__gt__` (`pyrad.io.read_data_mxpole.pyrad_MCH` `attribute`), 109
`__gt__` (`pyrad.io.read_data_mxpole.pyrad_MXPOL` `attribute`), 115
`__gt__` (`pyrad.io.timeseries.TimeSeries` `attribute`), 150
`__gt__` (`pyrad.io.timeseries._DataSeries` `attribute`), 151
`__gt__` (`pyrad.io.trajectory.Trajectory` `attribute`), 154
`__gt__` (`pyrad.io.trajectory._Radar_Trajectory` `attribute`), 157
`__gt__` (`pyrad.proc.process_traj.TargetRadar` `attribute`), 62
`__hash__` (`pyrad.io.read_data_mxpole.pyrad_IDL` `attribute`), 104
`__hash__` (`pyrad.io.read_data_mxpole.pyrad_MCH` `attribute`), 109
`__hash__` (`pyrad.io.read_data_mxpole.pyrad_MXPOL` `attribute`), 115
`__hash__` (`pyrad.io.timeseries.TimeSeries` `attribute`), 150
`__hash__` (`pyrad.io.timeseries._DataSeries` `attribute`), 151
`__hash__` (`pyrad.io.trajectory.Trajectory` `attribute`), 154
`__hash__` (`pyrad.io.trajectory._Radar_Trajectory` `attribute`), 157
`__hash__` (`pyrad.proc.process_traj.TargetRadar` `attribute`), 62
`__init__()` (`pyrad.io.read_data_mxpole.pyrad_IDL` `method`), 104
`__init__()` (`pyrad.io.read_data_mxpole.pyrad_MCH` `method`), 109
`__init__()` (`pyrad.io.read_data_mxpole.pyrad_MXPOL` `method`), 115
`__init__()` (`pyrad.io.timeseries.TimeSeries` `method`), 150
`__init__()` (`pyrad.io.timeseries._DataSeries` `method`), 152
`__init__()` (`pyrad.io.trajectory.Trajectory` `method`), 154
`__init__()` (`pyrad.io.trajectory._Radar_Trajectory` `method`), 157
`__init__()` (`pyrad.proc.process_traj.TargetRadar` `method`), 62
`__init_subclass__()` (`pyrad.io.read_data_mxpole.pyrad_IDL` `method`), 104
`__init_subclass__()` (`pyrad.io.read_data_mxpole.pyrad_MCH` `method`), 109
`__init_subclass__()` (`pyrad.io.read_data_mxpole.pyrad_MXPOL` `method`), 115
`__init_subclass__()` (`pyrad.io.timeseries.TimeSeries` `method`), 150
`__init_subclass__()` (`pyrad.io.timeseries._DataSeries` `method`), 152

```

__init_subclass__()
    (pyrad.io.trajectory.Trajectory method), 154
__init_subclass__()
    (pyrad.io.trajectory._Radar_Trajectory method), 157
__init_subclass__()
    (pyrad.proc.process_traj.TargetRadar method), 62
__le__ (pyrad.io.read_data_mxpole.pyrad_IDL attribute), 104
__le__ (pyrad.io.read_data_mxpole.pyrad_MCH attribute), 109
__le__ (pyrad.io.read_data_mxpole.pyrad_MXPOL attribute), 115
__le__ (pyrad.io.timeseries.TimeSeries attribute), 150
__le__ (pyrad.io.timeseries._DataSeries attribute), 152
__le__ (pyrad.io.trajectory.Trajectory attribute), 154
__le__ (pyrad.io.trajectory._Radar_Trajectory attribute), 157
__le__ (pyrad.proc.process_traj.TargetRadar attribute), 62
__lt__ (pyrad.io.read_data_mxpole.pyrad_IDL attribute), 104
__lt__ (pyrad.io.read_data_mxpole.pyrad_MCH attribute), 109
__lt__ (pyrad.io.read_data_mxpole.pyrad_MXPOL attribute), 115
__lt__ (pyrad.io.timeseries.TimeSeries attribute), 150
__lt__ (pyrad.io.timeseries._DataSeries attribute), 152
__lt__ (pyrad.io.trajectory.Trajectory attribute), 155
__lt__ (pyrad.io.trajectory._Radar_Trajectory attribute), 157
__lt__ (pyrad.proc.process_traj.TargetRadar attribute), 62
__module__ (pyrad.io.read_data_mxpole.pyrad_IDL attribute), 104
__module__ (pyrad.io.read_data_mxpole.pyrad_MCH attribute), 109
__module__ (pyrad.io.read_data_mxpole.pyrad_MXPOL attribute), 115
__module__ (pyrad.io.timeseries.TimeSeries attribute), 150
__module__ (pyrad.io.timeseries._DataSeries attribute), 152
__module__ (pyrad.io.trajectory.Trajectory attribute), 155
__module__ (pyrad.io.trajectory._Radar_Trajectory attribute), 157
__module__ (pyrad.proc.process_traj.TargetRadar attribute), 62
__ne__ (pyrad.io.read_data_mxpole.pyrad_IDL attribute), 104
__ne__ (pyrad.io.read_data_mxpole.pyrad_MCH attribute), 109
__ne__ (pyrad.io.read_data_mxpole.pyrad_MXPOL attribute), 115
__ne__ (pyrad.io.timeseries.TimeSeries attribute), 150
__ne__ (pyrad.io.timeseries._DataSeries attribute), 152
__ne__ (pyrad.io.trajectory.Trajectory attribute), 155
__ne__ (pyrad.io.trajectory._Radar_Trajectory attribute), 157
__ne__ (pyrad.proc.process_traj.TargetRadar attribute), 62
__new__() (pyrad.io.read_data_mxpole.pyrad_IDL method), 104
__new__() (pyrad.io.read_data_mxpole.pyrad_MCH method), 109
__new__() (pyrad.io.read_data_mxpole.pyrad_MXPOL method), 115
__new__() (pyrad.io.timeseries.TimeSeries method), 150
__new__() (pyrad.io.timeseries._DataSeries method), 152
__new__() (pyrad.io.trajectory.Trajectory method), 155
__new__() (pyrad.io.trajectory._Radar_Trajectory method), 157
__new__() (pyrad.proc.process_traj.TargetRadar method), 62
__reduce__() (pyrad.io.read_data_mxpole.pyrad_IDL method), 104
__reduce__() (pyrad.io.read_data_mxpole.pyrad_MCH method), 110
__reduce__() (pyrad.io.read_data_mxpole.pyrad_MXPOL method), 115
__reduce__() (pyrad.io.timeseries.TimeSeries method), 150
__reduce__() (pyrad.io.timeseries._DataSeries method), 152
__reduce__() (pyrad.io.trajectory.Trajectory method), 155
__reduce__() (pyrad.io.trajectory._Radar_Trajectory method), 157
__reduce__() (pyrad.proc.process_traj.TargetRadar method), 62
__reduce_ex__() (pyrad.io.read_data_mxpole.pyrad_IDL method), 104
__reduce_ex__() (pyrad.io.read_data_mxpole.pyrad_MCH method), 110
__reduce_ex__() (pyrad.io.read_data_mxpole.pyrad_MXPOL method), 115
__reduce_ex__() (pyrad.io.timeseries.TimeSeries method), 150
__reduce_ex__() (pyrad.io.timeseries._DataSeries method), 152
__reduce_ex__() (pyrad.io.trajectory.Trajectory method), 155

```

<code>__reduce_ex__()</code> (<code>pyrad.io.trajectory._Radar_Trajectory</code> attribute), 157	<code>__sizeof__()</code> (<code>pyrad.io.trajectory._Radar_Trajectory</code> method), 157
<code>__reduce_ex__()</code> (<code>pyrad.proc.process_traj.TargetRadar</code> attribute), 62	<code>__sizeof__()</code> (<code>pyrad.proc.process_traj.TargetRadar</code> method), 62
<code>__repr__</code> (<code>pyrad.io.read_data_mxpole.pyrad_IDL</code> attribute), 104	<code>__str__</code> (<code>pyrad.io.read_data_mxpole.pyrad_IDL</code> attribute), 104
<code>__repr__</code> (<code>pyrad.io.read_data_mxpole.pyrad_MCH</code> attribute), 110	<code>__str__</code> (<code>pyrad.io.read_data_mxpole.pyrad_MCH</code> attribute), 110
<code>__repr__</code> (<code>pyrad.io.read_data_mxpole.pyrad_MXPOL</code> attribute), 115	<code>__str__</code> (<code>pyrad.io.read_data_mxpole.pyrad_MXPOL</code> attribute), 115
<code>__repr__</code> (<code>pyrad.io.timeseries.TimeSeries</code> attribute), 150	<code>__str__</code> (<code>pyrad.io.timeseries.TimeSeries</code> attribute), 150
<code>__repr__</code> (<code>pyrad.io.timeseries._DataSeries</code> attribute), 152	<code>__str__</code> (<code>pyrad.io.timeseries._DataSeries</code> attribute), 152
<code>__repr__</code> (<code>pyrad.io.trajectory.Trajectory</code> attribute), 155	<code>__str__</code> (<code>pyrad.io.trajectory.Trajectory</code> attribute), 155
<code>__repr__</code> (<code>pyrad.io.trajectory._Radar_Trajectory</code> attribute), 157	<code>__str__</code> (<code>pyrad.io.trajectory._Radar_Trajectory</code> attribute), 157
<code>__repr__</code> (<code>pyrad.proc.process_traj.TargetRadar</code> attribute), 62	<code>__str__</code> (<code>pyrad.proc.process_traj.TargetRadar</code> attribute), 62
<code>__setattr__</code> (<code>pyrad.io.read_data_mxpole.pyrad_IDL</code> attribute), 104	<code>__subclasshook__()</code> (<code>pyrad.io.read_data_mxpole.pyrad_IDL</code> method), 104
<code>__setattr__</code> (<code>pyrad.io.read_data_mxpole.pyrad_MCH</code> attribute), 110	<code>__subclasshook__()</code> (<code>pyrad.io.read_data_mxpole.pyrad_MCH</code> method), 110
<code>__setattr__</code> (<code>pyrad.io.read_data_mxpole.pyrad_MXPOL</code> attribute), 115	<code>__subclasshook__()</code> (<code>pyrad.io.read_data_mxpole.pyrad_MXPOL</code> method), 115
<code>__setattr__</code> (<code>pyrad.io.timeseries.TimeSeries</code> attribute), 150	<code>__subclasshook__()</code> (<code>pyrad.io.timeseries.TimeSeries</code> method), 150
<code>__setattr__</code> (<code>pyrad.io.timeseries._DataSeries</code> attribute), 152	<code>__subclasshook__()</code> (<code>pyrad.io.timeseries._DataSeries</code> method), 152
<code>__setattr__</code> (<code>pyrad.io.trajectory.Trajectory</code> attribute), 155	<code>__subclasshook__()</code> (<code>pyrad.io.trajectory.Trajectory</code> method), 155
<code>__setattr__</code> (<code>pyrad.io.trajectory._Radar_Trajectory</code> attribute), 157	<code>__subclasshook__()</code> (<code>pyrad.io.trajectory._Radar_Trajectory</code> method), 157
<code>__setattr__</code> (<code>pyrad.proc.process_traj.TargetRadar</code> attribute), 62	<code>__subclasshook__()</code> (<code>pyrad.proc.process_traj.TargetRadar</code> method), 62
<code>__setstate__()</code> (<code>pyrad.io.read_data_mxpole.pyrad_IDL</code> method), 104	<code>__weakref__</code> (<code>pyrad.io.read_data_mxpole.pyrad_IDL</code> attribute), 105
<code>__setstate__()</code> (<code>pyrad.io.read_data_mxpole.pyrad_MCH</code> method), 110	<code>__weakref__</code> (<code>pyrad.io.read_data_mxpole.pyrad_MCH</code> attribute), 110
<code>__setstate__()</code> (<code>pyrad.io.read_data_mxpole.pyrad_MXPOL</code> method), 115	<code>__weakref__</code> (<code>pyrad.io.read_data_mxpole.pyrad_MXPOL</code> attribute), 115
<code>__sizeof__()</code> (<code>pyrad.io.read_data_mxpole.pyrad_IDL</code> method), 104	<code>__weakref__</code> (<code>pyrad.io.timeseries.TimeSeries</code> attribute), 151
<code>__sizeof__()</code> (<code>pyrad.io.read_data_mxpole.pyrad_MCH</code> method), 110	<code>__weakref__</code> (<code>pyrad.io.timeseries._DataSeries</code> attribute), 152
<code>__sizeof__()</code> (<code>pyrad.io.read_data_mxpole.pyrad_MXPOL</code> method), 115	<code>__weakref__</code> (<code>pyrad.io.trajectory.Trajectory</code> attribute), 155
<code>__sizeof__()</code> (<code>pyrad.io.timeseries.TimeSeries</code> method), 150	
<code>__sizeof__()</code> (<code>pyrad.io.timeseries._DataSeries</code> method), 152	
<code>__sizeof__()</code> (<code>pyrad.io.trajectory.Trajectory</code> method), 155	

`__weakref__` (*pyrad.io.trajectory.Radar_Trajectory* attribute), 158
`__weakref__` (*pyrad.proc.process_traj.TargetRadar* attribute), 62
`_add_dataset()` (in module *pyrad.flow.flow_aux*), 3
`_check_sweep_in_range()` (*pyrad.io.read_data_mxpole.pyrad_IDL* method), 105
`_check_sweep_in_range()` (*pyrad.io.read_data_mxpole.pyrad_MCH* method), 110
`_check_sweep_in_range()` (*pyrad.io.read_data_mxpole.pyrad_MXPOL* method), 115
`_convert_traj_to_swissgrid()` (*pyrad.io.trajectory.Trajectory* method), 155
`_create_cfg_dict()` (in module *pyrad.flow.flow_aux*), 3
`_create_datacfg_dict()` (in module *pyrad.flow.flow_aux*), 4
`_create_dscfg_dict()` (in module *pyrad.flow.flow_aux*), 4
`_create_prdcfg_dict()` (in module *pyrad.flow.flow_aux*), 4
`_dic_info()` (*pyrad.io.read_data_mxpole.pyrad_IDL* method), 105
`_dic_info()` (*pyrad.io.read_data_mxpole.pyrad_MCH* method), 110
`_dic_info()` (*pyrad.io.read_data_mxpole.pyrad_MXPOL* method), 115
`_generate_dataset()` (in module *pyrad.flow.flow_aux*), 4
`_generate_prod()` (in module *pyrad.flow.flow_aux*), 4
`_get_closests_bin()` (in module *pyrad.proc.process_traj*), 62
`_get_datasets_list()` (in module *pyrad.flow.flow_aux*), 4
`_get_datatype_list()` (in module *pyrad.flow.flow_aux*), 5
`_get_datetime()` (in module *pyrad.io.io_aux*), 85
`_get_gates()` (in module *pyrad.proc.process_traj*), 63
`_get_gates_antenna_pattern()` (in module *pyrad.proc.process_traj*), 63
`_get_gates_trt()` (in module *pyrad.proc.process_traj*), 64
`_get_masterfile_list()` (in module *pyrad.flow.flow_aux*), 5
`_get_radars_data()` (in module *pyrad.flow.flow_aux*), 5
`_get_times_and_traj()` (in module *pyrad.flow.flow_aux*), 5
`_get_total_seconds()` (*pyrad.io.trajectory.Trajectory* method), 155
`_get_ts_values_antenna_pattern()` (in module *pyrad.proc.process_traj*), 64
`_initialize_datasets()` (in module *pyrad.flow.flow_aux*), 5
`_initialize_listener()` (in module *pyrad.flow.flow_aux*), 6
`_ncvar_to_dict()` (in module *pyrad.io.read_data_cosmo*), 121
`_plot_time_range()` (in module *pyrad.graph.plots_vol*), 165
`_postprocess_datasets()` (in module *pyrad.flow.flow_aux*), 6
`_prepare_for_interpolation()` (in module *pyrad.io.read_data_cosmo*), 121
`_prepare_for_interpolation()` (in module *pyrad.io.read_data_hzt*), 125
`_process_datasets()` (in module *pyrad.flow.flow_aux*), 6
`_put_radar_in_swiss_coord()` (in module *pyrad.io.read_data_cosmo*), 122
`_read_traj()` (*pyrad.io.trajectory.Trajectory* method), 155
`_read_traj_lightning()` (*pyrad.io.trajectory.Trajectory* method), 155
`_read_traj_trt()` (*pyrad.io.trajectory.Trajectory* method), 155
`_sample_out_of_sector()` (in module *pyrad.proc.process_traj*), 64
`_user_input_listener()` (in module *pyrad.flow.flow_aux*), 6
`_wait_for_files()` (in module *pyrad.flow.flow_aux*), 6
`_wait_for_rainbow_datatypes()` (in module *pyrad.flow.flow_aux*), 7
`_warning_format()` (in module *pyrad.flow.flow_aux*), 7

A

`add_dataseriers()` (*pyrad.io.timeseries.TimeSeries* method), 151
`add_field()` (in module *pyrad.io.read_data_radar*), 93
`add_field()` (*pyrad.io.read_data_mxpole.pyrad_IDL* method), 105
`add_field()` (*pyrad.io.read_data_mxpole.pyrad_MCH* method), 110
`add_field()` (*pyrad.io.read_data_mxpole.pyrad_MXPOL* method), 116
`add_field_like()` (*pyrad.io.read_data_mxpole.pyrad_IDL* method), 105

add_field_like() (pyrad.io.read_data_mxpole.pyrad_MCH method), 110
 add_field_like() (pyrad.io.read_data_mxpole.pyrad_MXPOL method), 116
 add_radar() (pyrad.io.trajectory.Trajectory method), 155
 add_timesample() (pyrad.io.timeseries.TimeSeries method), 151
 assign_trajectory() (pyrad.io.trajectory._Radar_Trajectory method), 158
 assign_velocity_vecs() (pyrad.io.trajectory._Radar_Trajectory method), 158

B

belongs_roi_indices() (in module pyrad.util.radar_utils), 180

C

calculate_velocities() (pyrad.io.trajectory.Trajectory method), 156
 check_field_exists() (pyrad.io.read_data_mxpole.pyrad_IDL method), 105
 check_field_exists() (pyrad.io.read_data_mxpole.pyrad_MCH method), 111
 check_field_exists() (pyrad.io.read_data_mxpole.pyrad_MXPOL method), 116
 compute_1d_stats() (in module pyrad.util.radar_utils), 180
 compute_2d_hist() (in module pyrad.util.radar_utils), 180
 compute_2d_stats() (in module pyrad.util.radar_utils), 180
 compute_directional_stats() (in module pyrad.util.radar_utils), 181
 compute_histogram() (in module pyrad.util.radar_utils), 181
 compute_histogram_sweep() (in module pyrad.util.radar_utils), 181
 compute_profile_stats() (in module pyrad.util.radar_utils), 181
 compute_quantiles() (in module pyrad.util.radar_utils), 182
 compute_quantiles_from_hist() (in module pyrad.util.radar_utils), 182
 compute_quantiles_sweep() (in module pyrad.util.radar_utils), 183
 convert_polvar_name() (in module pyrad.io.read_data_mxpole), 101

convert_radpos_to_swissgrid() (pyrad.io.trajectory._Radar_Trajectory method), 158
 cosmo2radar_coord() (in module pyrad.io.read_data_cosmo), 122
 cosmo2radar_data() (in module pyrad.io.read_data_cosmo), 122
 create_sun_hits_field() (in module pyrad.util.radar_utils), 183
 create_sun_retrieval_field() (in module pyrad.util.radar_utils), 183

E

extract_sweeps() (pyrad.io.read_data_mxpole.pyrad_IDL method), 105
 extract_sweeps() (pyrad.io.read_data_mxpole.pyrad_MCH method), 111
 extract_sweeps() (pyrad.io.read_data_mxpole.pyrad_MXPOL method), 116

F

find_colocated_indexes() (in module pyrad.util.radar_utils), 183
 find_cosmo_file() (in module pyrad.io.io_aux), 86
 find_hzt_file() (in module pyrad.io.io_aux), 86
 find_nearest_gate() (in module pyrad.util.radar_utils), 184
 find_neighbour_gates() (in module pyrad.util.radar_utils), 184
 find_rad4alpcosmo_file() (in module pyrad.io.io_aux), 86
 find_raw_cosmo_file() (in module pyrad.io.io_aux), 86
 find_ray_index() (in module pyrad.util.radar_utils), 184
 find_rng_index() (in module pyrad.util.radar_utils), 184
 findTimes() (in module pyrad.io.read_data_mxpole), 101

G

generate_colocated_gates_products() (in module pyrad.prod.process_intercomp_products), 81
 generate_cosmo_coord_products() (in module pyrad.prod.process_product), 71
 generate_field_name_str() (in module pyrad.io.io_aux), 87
 generate_grid_products() (in module pyrad.prod.process_grid_products), 75
 generate_intercomp_products() (in module pyrad.prod.process_intercomp_products), 81
 generate_ml_products() (in module pyrad.prod.process_product), 71

`generate_monitoring_products()` (in module `pyrad.prod.process_monitoring_products`), 79
`generate_occurrence_products()` (in module `pyrad.prod.process_product`), 71
`generate_polvar_metadata()` (in module `pyrad.io.read_data_mxp`), 101
`generate_qvp_products()` (in module `pyrad.prod.process_product`), 71
`generate_radar_table()` (in module `pyrad.io.read_data_mxp`), 102
`generate_sparse_grid_products()` (in module `pyrad.prod.process_grid_products`), 75
`generate_sun_hits_products()` (in module `pyrad.prod.process_product`), 72
`generate_time_avg_products()` (in module `pyrad.prod.process_intercomp_products`), 81
`generate_timeseries_products()` (in module `pyrad.prod.process_timeseries_products`), 77
`generate_traj_product()` (in module `pyrad.prod.process_traj_products`), 83
`generate_vol_products()` (in module `pyrad.prod.process_vol_products`), 73
`get_array()` (in module `pyrad.io.config`), 91
`get_array_type()` (in module `pyrad.io.config`), 91
`get_azimuth()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 106
`get_azimuth()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 111
`get_azimuth()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 116
`get_closest_solar_flux()` (in module `pyrad.util.radar_utils`), 185
`get_colobar_label()` (in module `pyrad.graph.plots_aux`), 159
`get_cosmo_fields()` (in module `pyrad.io.read_data_cosmo`), 122
`get_data()` (in module `pyrad.io.read_data_radar`), 93
`get_data_mxp()` (in module `pyrad.io.read_data_radar`), 94
`get_data_odim()` (in module `pyrad.io.read_data_radar`), 94
`get_data_rad4alp()` (in module `pyrad.io.read_data_radar`), 94
`get_data_rainbow()` (in module `pyrad.io.read_data_radar`), 94
`get_dataset_fields()` (in module `pyrad.io.io_aux`), 87
`get_datatype_fields()` (in module `pyrad.io.io_aux`), 87
`get_datatype_metranet()` (in module `pyrad.io.io_aux`), 87
`get_datatype_odim()` (in module `pyrad.io.io_aux`), 88
`get_datetime()` (in module `pyrad.io.io_aux`), 88
`get_elevation()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 106
`get_elevation()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 111
`get_elevation()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 117
`get_elevation_metadata()` (in module `pyrad.io.read_data_mxp`), 102
`get_end()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 106
`get_end()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 111
`get_end()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 117
`get_end_time()` (`pyrad.io.trajectory.Trajectory` method), 156
`get_field()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 106
`get_field()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 112
`get_field()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 117
`get_field_name()` (in module `pyrad.graph.plots_aux`), 159
`get_field_name()` (in module `pyrad.io.io_aux`), 88
`get_field_unit()` (in module `pyrad.io.io_aux`), 88
`get_fieldname_cosmo()` (in module `pyrad.io.io_aux`), 88
`get_fieldname_pyart()` (in module `pyrad.io.io_aux`), 88
`get_file_list()` (in module `pyrad.io.io_aux`), 89
`get_gate_x_y_z()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 106
`get_gate_x_y_z()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 112
`get_gate_x_y_z()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 117
`get_histogram_bins()` (in module `pyrad.util.radar_utils`), 185
`get_iso0_field()` (in module `pyrad.io.read_data_hzt`), 125
`get_mymetadata()` (in module `pyrad.io.read_data_mxp`), 102
`get_new_rainbow_file_name()` (in module `pyrad.io.io_aux`), 89
`get_norm()` (in module `pyrad.graph.plots_aux`), 159
`get_num_elements()` (in module `pyrad.io.config`), 91
`get_nyquist_vel()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 107
`get_nyquist_vel()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 112

`get_nyquist_vel()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`get_process_func()` (in module `pyrad.proc.process_aux`), 11

`get_prodcen_func()` (in module `pyrad.prod.product_aux`), 69

`get_range_bins_to_avg()` (in module `pyrad.util.radar_utils`), 185

`get_ROI()` (in module `pyrad.util.radar_utils`), 185

`get_samples_in_period()` (`pyrad.io.trajectory.Trajectory` method), 156

`get_save_dir()` (in module `pyrad.io.io_aux`), 89

`get_scan_list()` (in module `pyrad.io.io_aux`), 89

`get_sensor_data()` (in module `pyrad.io.read_data_sensor`), 127

`get_slice()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 107

`get_slice()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`get_slice()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`get_start()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 107

`get_start()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`get_start()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`get_start_end()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 107

`get_start_end()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`get_start_end()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`get_start_time()` (`pyrad.io.trajectory.Trajectory` method), 156

`get_struct()` (in module `pyrad.io.config`), 92

`get_target_elevations()` (in module `pyrad.util.radar_utils`), 185

`get_trtfile_list()` (in module `pyrad.io.io_aux`), 90

H

`hzt2radar_coord()` (in module `pyrad.io.read_data_hzt`), 126

`hzt2radar_data()` (in module `pyrad.io.read_data_hzt`), 126

I

`info()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 107

`info()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`info()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`init_array()` (in module `pyrad.io.config`), 92

`init_gate_altitude()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 107

`init_gate_altitude()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`init_gate_altitude()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`init_gate_longitude_latitude()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 107

`init_gate_longitude_latitude()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`init_gate_longitude_latitude()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`init_gate_x_y_z()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 107

`init_gate_x_y_z()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`init_gate_x_y_z()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`init_rays_per_sweep()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 108

`init_rays_per_sweep()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`init_rays_per_sweep()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`int2float_radar()` (in module `pyrad.io.read_data_mxp`), 102

`interpol_field()` (in module `pyrad.io.read_data_radar`), 95

`iter_azimuth()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 108

`iter_azimuth()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`iter_azimuth()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`iter_elevation()` (`pyrad.io.read_data_mxp.pyrad_IDL` method), 108

`iter_elevation()` (`pyrad.io.read_data_mxp.pyrad_MCH` method), 113

`iter_elevation()` (`pyrad.io.read_data_mxp.pyrad_MXPOL` method), 118

`method`), 118
`iter_end()` (`pyrad.io.read_data_mxpole.pyrad_IDL`
`method`), 108
`iter_end()` (`pyrad.io.read_data_mxpole.pyrad_MCH`
`method`), 113
`iter_end()` (`pyrad.io.read_data_mxpole.pyrad_MXPOL`
`method`), 119
`iter_field()` (`pyrad.io.read_data_mxpole.pyrad_IDL`
`method`), 108
`iter_field()` (`pyrad.io.read_data_mxpole.pyrad_MCH`
`method`), 113
`iter_field()` (`pyrad.io.read_data_mxpole.pyrad_MXPOL`
`method`), 119
`iter_slice()` (`pyrad.io.read_data_mxpole.pyrad_IDL`
`method`), 108
`iter_slice()` (`pyrad.io.read_data_mxpole.pyrad_MCH`
`method`), 113
`iter_slice()` (`pyrad.io.read_data_mxpole.pyrad_MXPOL`
`method`), 119
`iter_start()` (`pyrad.io.read_data_mxpole.pyrad_IDL`
`method`), 108
`iter_start()` (`pyrad.io.read_data_mxpole.pyrad_MCH`
`method`), 113
`iter_start()` (`pyrad.io.read_data_mxpole.pyrad_MXPOL`
`method`), 119
`iter_start_end()` (`pyrad.io.read_data_mxpole.pyrad_IDL`
`method`), 108
`iter_start_end()` (`pyrad.io.read_data_mxpole.pyrad_MCH`
`method`), 113
`iter_start_end()` (`pyrad.io.read_data_mxpole.pyrad_MXPOL`
`method`), 119

J

`join_time_series()` (in module `pyrad.util.radar_utils`), 186

L

`load_myconfig()` (in module `pyrad.io.read_data_mxpole`), 102
`location_is_equal()` (`pyrad.io.trajectory._Radar_Trajectory`
`method`), 158

M

`main()` (in module `pyrad.flow.flow_control`), 9
`main_rt()` (in module `pyrad.flow.flow_control`), 9
`make_filename()` (in module `pyrad.io.io_aux`), 90
`map_Doppler()` (in module `pyrad.io.io_aux`), 90
`map_hydro()` (in module `pyrad.io.io_aux`), 90
`merge_fields_cfradial()` (in module `pyrad.io.read_data_radar`), 95
`merge_fields_cosmo()` (in module `pyrad.io.read_data_radar`), 95
`merge_fields_dem()` (in module `pyrad.io.read_data_radar`), 95
`merge_fields_rainbow()` (in module `pyrad.io.read_data_radar`), 96
`merge_scans_cosmo()` (in module `pyrad.io.read_data_radar`), 96
`merge_scans_cosmo_rad4alp()` (in module `pyrad.io.read_data_radar`), 96
`merge_scans_dem()` (in module `pyrad.io.read_data_radar`), 97
`merge_scans_dem_rad4alp()` (in module `pyrad.io.read_data_radar`), 97
`merge_scans_Doppler_rad4alp()` (in module `pyrad.io.read_data_radar`), 96
`merge_scans_hydro_rad4alp()` (in module `pyrad.io.read_data_radar`), 97
`merge_scans_mxpole()` (in module `pyrad.io.read_data_radar`), 97
`merge_scans_odim()` (in module `pyrad.io.read_data_radar`), 98
`merge_scans_rad4alp()` (in module `pyrad.io.read_data_radar`), 98
`merge_scans_rainbow()` (in module `pyrad.io.read_data_radar`), 98
`plot()` (`pyrad.io.timeseries.TimeSeries` `method`), 151
`plot_along_coord()` (in module `pyrad.graph.plots_vol`), 166
`plot_antenna_pattern()` (in module `pyrad.graph.plots`), 161
`plot_bscope()` (in module `pyrad.graph.plots_vol`), 166
`plot_cappi()` (in module `pyrad.graph.plots_vol`), 166
`plot_density()` (in module `pyrad.graph.plots`), 161
`plot_field_coverage()` (in module `pyrad.graph.plots_vol`), 167
`plot_hist()` (`pyrad.io.timeseries.TimeSeries`
`method`), 151
`plot_histogram()` (in module `pyrad.graph.plots`), 162
`plot_histogram2()` (in module `pyrad.graph.plots`), 162
`plot_intercomp_scores_ts()` (in module `pyrad.graph.plots_timeseries`), 175
`plot_latitude_slice()` (in module `pyrad.graph.plots_grid`), 173
`plot_latlon_slice()` (in module `pyrad.graph.plots_grid`), 173
`plot_longitude_slice()` (in module `pyrad.graph.plots_grid`), 173
`plot_ml_ts()` (in module `pyrad.graph.plots_timeseries`), 176

<code>plot_monitoring_ts()</code>	(in module <code>pyrad.graph.plots_timeseries</code>), 176	<code>pyrad.proc.process_cosmo</code>), 19
<code>plot_pos()</code>	(in module <code>pyrad.graph.plots_vol</code>), 167	<code>pyrad.proc.process_cosmo</code>), 19
<code>plot_ppi()</code>	(in module <code>pyrad.graph.plots_vol</code>), 168	<code>pyrad.proc.process_cosmo_lookup_table()</code>
<code>plot_ppi_contour()</code>	(in module <code>pyrad.graph.plots_vol</code>), 168	(in module <code>pyrad.proc.process_cosmo</code>), 20
<code>plot_ppi_map()</code>	(in module <code>pyrad.graph.plots_vol</code>), 169	<code>process_dealias_fourdd()</code>
<code>plot_quantiles()</code>	(in module <code>pyrad.graph.plots</code>), 163	(in module <code>pyrad.proc.process_Doppler</code>), 23
<code>plot_rhi()</code>	(in module <code>pyrad.graph.plots_vol</code>), 169	<code>process_dealias_region_based()</code>
<code>plot_rhi_contour()</code>	(in module <code>pyrad.graph.plots_vol</code>), 169	(in module <code>pyrad.proc.process_Doppler</code>), 23
<code>plot_rhi_profile()</code>	(in module <code>pyrad.graph.plots_vol</code>), 170	<code>process_dealias_unwrap_phase()</code>
<code>plot_scatter()</code>	(in module <code>pyrad.graph.plots</code>), 163	(in module <code>pyrad.proc.process_Doppler</code>), 24
<code>plot_scatter_comp()</code>	(in module <code>pyrad.graph.plots</code>), 163	<code>process_echo_filter()</code>
<code>plot_sun_hits()</code>	(in module <code>pyrad.graph.plots</code>), 164	(in module <code>pyrad.proc.process_echoclass</code>), 28
<code>plot_sun_retrieval_ts()</code>	(in module <code>pyrad.graph.plots_timeseries</code>), 177	<code>process_echo_id()</code>
<code>plot_surface()</code>	(in module <code>pyrad.graph.plots_grid</code>), 174	(in module <code>pyrad.proc.process_echoclass</code>), 28
<code>plot_time_range()</code>	(in module <code>pyrad.graph.plots_vol</code>), 170	<code>process_estimate_phidp0()</code>
<code>plot_timeseries()</code>	(in module <code>pyrad.graph.plots_timeseries</code>), 177	(in module <code>pyrad.proc.process_monitoring</code>), 37
<code>plot_timeseries_comp()</code>	(in module <code>pyrad.graph.plots_timeseries</code>), 178	<code>process_evdp()</code>
<code>plot_traj()</code>	(in module <code>pyrad.graph.plots_vol</code>), 171	(in module <code>pyrad.proc.process_timeseries</code>), 55
<code>process_attenuation()</code>	(in module <code>pyrad.proc.process_phase</code>), 43	<code>process_filter_snr()</code>
<code>process_azimuthal_average()</code>	(in module <code>pyrad.proc.process_aux</code>), 11	(in module <code>pyrad.proc.process_echoclass</code>), 29
<code>process_bird_density()</code>	(in module <code>pyrad.proc.process_retrieve</code>), 49	<code>process_filter_vel_diff()</code>
<code>process_birds_id()</code>	(in module <code>pyrad.proc.process_echoclass</code>), 27	(in module <code>pyrad.proc.process_echoclass</code>), 29
<code>process_cdf()</code>	(in module <code>pyrad.proc.process_echoclass</code>), 27	<code>process_filter_visibility()</code>
<code>process_cdr()</code>	(in module <code>pyrad.proc.process_retrieve</code>), 49	(in module <code>pyrad.proc.process_echoclass</code>), 29
<code>process_clt_to_echo_id()</code>	(in module <code>pyrad.proc.process_echoclass</code>), 28	<code>process_gc_monitoring()</code>
<code>process_colocated_gates()</code>	(in module <code>pyrad.proc.process_intercomp</code>), 33	(in module <code>pyrad.proc.process_calib</code>), 16
<code>process_correct_bias()</code>	(in module <code>pyrad.proc.process_calib</code>), 15	<code>process_grid()</code>
<code>process_correct_noise_rhohv()</code>	(in module <code>pyrad.proc.process_calib</code>), 15	(in module <code>pyrad.proc.process_aux</code>), 12
<code>process_correct_phidp0()</code>	(in module <code>pyrad.proc.process_phase</code>), 44	<code>process_hydroclass()</code>
<code>process_cosmo()</code>	(in module <code>pyrad.proc.process_cosmo</code>), 19	(in module <code>pyrad.proc.process_echoclass</code>), 29
		<code>process_hzt()</code>
		(in module <code>pyrad.proc.process_cosmo</code>), 20
		<code>process_hzt_coord()</code>
		(in module <code>pyrad.proc.process_cosmo</code>), 21
		<code>process_hzt_lookup_table()</code>
		(in module <code>pyrad.proc.process_cosmo</code>), 21
		<code>process_intercomp()</code>
		(in module <code>pyrad.proc.process_intercomp</code>), 34
		<code>process_intercomp_time_avg()</code>
		(in module <code>pyrad.proc.process_intercomp</code>), 34
		<code>process_kdp_least_square_double_window()</code>
		(in module <code>pyrad.proc.process_phase</code>), 44
		<code>process_kdp_least_square_single_window()</code>
		(in module <code>pyrad.proc.process_phase</code>), 45
		<code>process_l()</code>
		(in module <code>pyrad.proc.process_retrieve</code>), 50
		<code>process_melting_layer()</code>
		(in module <code>pyrad.proc.process_echoclass</code>), 30
		<code>process_monitoring()</code>
		(in module <code>pyrad.proc.process_monitoring</code>), 37
		<code>process_occurrence()</code>
		(in module <code>pyrad.proc.process_calib</code>), 16
		<code>process_occurrence_period()</code>
		(in module <code>pyrad.proc.process_calib</code>), 16

pyrad.proc.process_calib), 17
process_outlier_filter() (in module *pyrad.proc.process_echoclass*), 30
process_phidp_kdp_Kalman() (in module *pyrad.proc.process_phase*), 45
process_phidp_kdp_lp() (in module *pyrad.proc.process_phase*), 46
process_phidp_kdp_Maesaka() (in module *pyrad.proc.process_phase*), 45
process_phidp_kdp_Vulpiani() (in module *pyrad.proc.process_phase*), 46
process_point_measurement() (in module *pyrad.proc.process_timeseries*), 56
process_qvp() (in module *pyrad.proc.process_timeseries*), 56
process_rainrate() (in module *pyrad.proc.process_retrieve*), 50
process_raw() (in module *pyrad.proc.process_aux*), 12
process_rcs() (in module *pyrad.proc.process_retrieve*), 51
process_rcs_pr() (in module *pyrad.proc.process_retrieve*), 51
process_rhohv_rain() (in module *pyrad.proc.process_monitoring*), 38
process_roi() (in module *pyrad.proc.process_aux*), 13
process_rqvp() (in module *pyrad.proc.process_timeseries*), 57
process_save_radar() (in module *pyrad.proc.process_aux*), 13
process_selfconsistency_bias() (in module *pyrad.proc.process_monitoring*), 38
process_selfconsistency_kdp_phidp() (in module *pyrad.proc.process_monitoring*), 39
process_signal_power() (in module *pyrad.proc.process_retrieve*), 52
process_smooth_phidp_double_window() (in module *pyrad.proc.process_phase*), 47
process_smooth_phidp_single_window() (in module *pyrad.proc.process_phase*), 47
process_snr() (in module *pyrad.proc.process_retrieve*), 53
process_sun_hits() (in module *pyrad.proc.process_calib*), 17
process_svp() (in module *pyrad.proc.process_timeseries*), 58
process_time_avg() (in module *pyrad.proc.process_intercomp*), 35
process_time_avg_flag() (in module *pyrad.proc.process_intercomp*), 35
process_time_height() (in module *pyrad.proc.process_timeseries*), 59
process_traj_antenna_pattern() (in module *pyrad.proc.process_traj*), 64
process_traj_atplane() (in module *pyrad.proc.process_traj*), 66
process_traj_lightning() (in module *pyrad.proc.process_traj*), 66
process_traj_trt() (in module *pyrad.proc.process_traj*), 66
process_trajectory() (in module *pyrad.proc.process_traj*), 67
process_vad() (in module *pyrad.proc.process_Doppler*), 24
process_vol_refl() (in module *pyrad.proc.process_retrieve*), 53
process_weighted_time_avg() (in module *pyrad.proc.process_intercomp*), 36
process_wind_vel() (in module *pyrad.proc.process_Doppler*), 25
process_windshear() (in module *pyrad.proc.process_Doppler*), 25
process_zdr_column() (in module *pyrad.proc.process_echoclass*), 31
process_zdr_precip() (in module *pyrad.proc.process_monitoring*), 39
process_zdr_snow() (in module *pyrad.proc.process_monitoring*), 40
profiler() (in module *pyrad.flow.flow_aux*), 7
project_to_vertical() (in module *pyrad.util.radar_utils*), 186
pyrad.flow.flow_aux (module), 1
pyrad.flow.flow_control (module), 7
pyrad.graph.plots (module), 159
pyrad.graph.plots_aux (module), 158
pyrad.graph.plots_grid (module), 171
pyrad.graph.plots_timeseries (module), 174
pyrad.graph.plots_vol (module), 164
pyrad.io.config (module), 90
pyrad.io.io_aux (module), 83
pyrad.io.mxpole_config (module), 120
pyrad.io.read_data_cosmo (module), 120
pyrad.io.read_data_hzt (module), 123
pyrad.io.read_data_mxpole (module), 99
pyrad.io.read_data_other (module), 134
pyrad.io.read_data_radar (module), 92
pyrad.io.read_data_sensor (module), 126
pyrad.io.read_data_sun (module), 131
pyrad.io.timeseries (module), 148
pyrad.io.trajectory (module), 152
pyrad.io.write_data (module), 140
pyrad.proc.process_aux (module), 10
pyrad.proc.process_calib (module), 13
pyrad.proc.process_cosmo (module), 18
pyrad.proc.process_Doppler (module), 21
pyrad.proc.process_echoclass (module), 25
pyrad.proc.process_intercomp (module), 31

pyrad.proc.process_monitoring (module), 36
 pyrad.proc.process_phase (module), 41
 pyrad.proc.process_retrieve (module), 47
 pyrad.proc.process_timeseries (module), 53
 pyrad.proc.process_traj (module), 59
 pyrad.prod.process_grid_products (module), 73
 pyrad.prod.process_intercomp_products (module), 79
 pyrad.prod.process_monitoring_products (module), 77
 pyrad.prod.process_product (module), 69
 pyrad.prod.process_timeseries_products (module), 75
 pyrad.prod.process_traj_products (module), 82
 pyrad.prod.process_vol_products (module), 72
 pyrad.prod.product_aux (module), 67
 pyrad.util.radar_utils (module), 178
 pyrad.util.stat_utils (module), 187
 pyrad_IDL (class in pyrad.io.read_data_mxp), 102
 pyrad_MCH (class in pyrad.io.read_data_mxp), 108
 pyrad_MXPOL (class in pyrad.io.read_data_mxp), 113

Q

quantiles_weighted() (in module pyrad.util.stat_utils), 189
 quantize_field() (in module pyrad.util.radar_utils), 186

R

rainfall_accumulation() (in module pyrad.util.radar_utils), 186
 read_antenna_pattern() (in module pyrad.io.read_data_other), 135
 read_colocated_data() (in module pyrad.io.read_data_other), 136
 read_colocated_data_time_avg() (in module pyrad.io.read_data_other), 136
 read_colocated_gates() (in module pyrad.io.read_data_other), 136
 read_config() (in module pyrad.io.config), 92
 read_cosmo_coord() (in module pyrad.io.read_data_cosmo), 123
 read_cosmo_data() (in module pyrad.io.read_data_cosmo), 123
 read_disdro() (in module pyrad.io.read_data_sensor), 127
 read_disdro_scattering() (in module pyrad.io.read_data_sensor), 128
 read_excess_gates() (in module pyrad.io.read_data_other), 136

read_histogram() (in module pyrad.io.read_data_other), 137
 read_histogram_ts() (in module pyrad.io.read_data_other), 137
 read_hzt_data() (in module pyrad.io.read_data_hzt), 126
 read_intercomp_scores_ts() (in module pyrad.io.read_data_other), 137
 read_intercomp_scores_ts_old() (in module pyrad.io.read_data_other), 137
 read_intercomp_scores_ts_old_v0() (in module pyrad.io.read_data_other), 137
 read_last_state() (in module pyrad.io.read_data_other), 138
 read_lightning() (in module pyrad.io.read_data_sensor), 128
 read_lightning_all() (in module pyrad.io.read_data_sensor), 128
 read_lightning_traj() (in module pyrad.io.read_data_sensor), 128
 read_meteorage() (in module pyrad.io.read_data_sensor), 129
 read_ml_ts() (in module pyrad.io.read_data_other), 138
 read_monitoring_ts() (in module pyrad.io.read_data_other), 138
 read_monitoring_ts_old() (in module pyrad.io.read_data_other), 138
 read_profile_ts() (in module pyrad.io.read_data_other), 138
 read_quantiles() (in module pyrad.io.read_data_other), 139
 read_quantiles_ts() (in module pyrad.io.read_data_other), 139
 read_rad4alp_cosmo() (in module pyrad.io.read_data_other), 139
 read_rad4alp_vis() (in module pyrad.io.read_data_other), 139
 read_rhi_profile() (in module pyrad.io.read_data_other), 139
 read_selfconsistency() (in module pyrad.io.read_data_other), 140
 read_smn() (in module pyrad.io.read_data_sensor), 129
 read_smn2() (in module pyrad.io.read_data_sensor), 129
 read_solar_flux() (in module pyrad.io.read_data_sun), 133
 read_status() (in module pyrad.io.read_data_other), 140
 read_sun_hits() (in module pyrad.io.read_data_sun), 133
 read_sun_hits_multiple_days() (in module pyrad.io.read_data_sun), 133

`read_sun_retrieval()` (in module `pyrad.io.read_data_sun`), 134
`read_timeseries()` (in module `pyrad.io.read_data_other`), 140
`read_trt_cell_lightning()` (in module `pyrad.io.read_data_sensor`), 129
`read_trt_data()` (in module `pyrad.io.read_data_sensor`), 129
`read_trt_scores()` (in module `pyrad.io.read_data_sensor`), 130
`read_trt_traj_data()` (in module `pyrad.io.read_data_sensor`), 130
`read_ts_cum()` (in module `pyrad.io.read_data_other`), 140
`readCHRadData()` (in module `pyrad.io.read_data_mxpolf`), 119
`readIDLRadData()` (in module `pyrad.io.read_data_mxpolf`), 119
`readMXPOLRadData()` (in module `pyrad.io.read_data_mxpolf`), 119
`row_stack()` (in module `pyrad.io.read_data_mxpolf`), 119

S

`send_msg()` (in module `pyrad.io.write_data`), 141
`set_value()` (`pyrad.io.timeseries.DataSeries` method), 152
`string_to_datatype()` (in module `pyrad.io.config`), 92

T

`TargetRadar` (class in `pyrad.proc.process_traj`), 61
`time_avg_range()` (in module `pyrad.util.radar_utils`), 187
`time_series_statistics()` (in module `pyrad.util.radar_utils`), 187
`TimeSeries` (class in `pyrad.io.timeseries`), 149
`Trajectory` (class in `pyrad.io.trajectory`), 153

W

`write()` (`pyrad.io.timeseries.TimeSeries` method), 151
`write_alarm_msg()` (in module `pyrad.io.write_data`), 142
`write_cdf()` (in module `pyrad.io.write_data`), 142
`write_colocated_data()` (in module `pyrad.io.write_data`), 142
`write_colocated_data_time_avg()` (in module `pyrad.io.write_data`), 143
`write_colocated_gates()` (in module `pyrad.io.write_data`), 143
`write_excess_gates()` (in module `pyrad.io.write_data`), 143
`write_field_coverage()` (in module `pyrad.io.write_data`), 143

`write_histogram()` (in module `pyrad.io.write_data`), 144
`write_intercomp_scores_ts()` (in module `pyrad.io.write_data`), 144
`write_last_state()` (in module `pyrad.io.write_data`), 144
`write_monitoring_ts()` (in module `pyrad.io.write_data`), 144
`write_quantiles()` (in module `pyrad.io.write_data`), 145
`write_rhi_profile()` (in module `pyrad.io.write_data`), 145
`write_smn()` (in module `pyrad.io.write_data`), 145
`write_sun_hits()` (in module `pyrad.io.write_data`), 146
`write_sun_retrieval()` (in module `pyrad.io.write_data`), 146
`write_trt_cell_data()` (in module `pyrad.io.write_data`), 146
`write_trt_cell_lightning()` (in module `pyrad.io.write_data`), 146
`write_trt_cell_scores()` (in module `pyrad.io.write_data`), 147
`write_ts_cum()` (in module `pyrad.io.write_data`), 147
`write_ts_lightning()` (in module `pyrad.io.write_data`), 147
`write_ts_ml()` (in module `pyrad.io.write_data`), 148
`write_ts_polar_data()` (in module `pyrad.io.write_data`), 148