pyrad library reference for users

Release 0.0.1

meteoswiss-mdr

Feb 02, 2018

Contents:

CHAPTER

ONE

PROCESSING FLOW CONTROL (PYRAD.FLOW)

Functions to control the Pyrad data processing flow

main(cfgfile[, starttime, endtime, ...])	main flow control. Processes radar data off-line over a period of time

pyrad.flow.**main**(*cfgfile*, *starttime=None*, *endtime=None*, *trajfile=''*, *trajtype='plane'*, *flashnr=0*, *infostr=''*)

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.

pyrad.flow.**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 inteded 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

CHAPTER

TWO

DATASET PROCESSING (PYRAD.PROC)

Initiate the dataset processing.

2.1 Auxiliary functions

get_process_func(dataset_type, dsname)	maps the dataset type into its processing function and data set format
process_raw(procstatus, dscfg[, radar_list])	dummy function that returns the initial input data set
process_save_radar(procstatus, dscfg[, ...])	dummy function that allows to save the entire radar object
process_point_measurement(procstatus, dscfg)	Obtains the radar data at a point measurement
process_grid(procstatus, dscfg[, radar_list])	Puts the radar data in a regular grid
process_qvp(procstatus, dscfg[, radar_list])	Computes quasi vertical profiles
process_time_height(procstatus, dscfg[, ...])	Produces time height radar objects at a point of interest defined by

2.2 Echo classification and filtering

process_echo_id(procstatus, dscfg[, radar_list])	identifies echoes as 0: No data, 1: Noise, 2: Clutter,
process_echo_filter(procstatus, dscfg[, ...])	Masks all echo types that are not of the class specified in
process_cdf(procstatus, dscfg[, radar_list])	Collects the fields necessary to compute the Cumulative Distribution
process_filter_snr(procstatus, dscfg[, ...])	filters out low SNR echoes
process_filter_visibility(procstatus, dscfg)	filters out rays gates with low visibility and corrects the reflectivity
process_outlier_filter(procstatus, dscfg[, ...])	filters out gates which are outliers respect to the surrounding
process_hydroclass(procstatus, dscfg[, ...])	Classifies precipitation echoes

2.3 Phase processing and attenuation correction

process_correct_phidp0(procstatus, dscfg[, ...])	corrects phidp of the system phase
process_smooth_phidp_single_window(...[, ...])	corrects phidp of the system phase and smoothes it using one window

Continued on next page

Table 2.3 – continued from previous page

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

2.4 Monitoring, calibration and noise correction

process_correct_bias(procstatus, dscfg[, ...])	Corrects a bias on the data
process_correct_noise_rhohv(procstatus, dscfg)	identifies echoes as 0: No data, 1: Noise, 2: Clutter,
process_rhohv_rain(procstatus, dscfg[, ...])	Keeps only suitable data to evaluate the 80 percentile of RhoHV in rain
process_zdr_precip(procstatus, dscfg[, ...])	Keeps only suitable data to evaluate the differential reflectivity in
process_zdr_snow(procstatus, dscfg[, radar_list])	Keeps only suitable data to evaluate the differential reflectivity in
process_estimate_phidp0(procstatus, dscfg[, ...])	estimates the system differential phase offset at each ray
process_sun_hits(procstatus, dscfg[, radar_list])	monitoring of the radar using sun hits
process_selfconsistency_kdp_phidp(...[, ...])	Computes specific differential phase and differential phase in rain using
process_selfconsistency_bias(procstatus, dscfg)	Estimates the reflectivity bias by means of the selfconsistency
process_monitoring(procstatus, dscfg[, ...])	computes monitoring statistics
process_time_avg(procstatus, dscfg[, radar_list])	computes the temporal mean of a field
process_weighted_time_avg(procstatus, dscfg)	computes the temporal mean of a field weighted by the reflectivity
process_time_avg_flag(procstatus, dscfg[, ...])	computes a flag field describing the conditions of the data used while
process_colocated_gates(procstatus, dscfg[, ...])	Find colocated gates within two radars
process_intercomp(procstatus, dscfg[, ...])	intercomparison between two radars
process_intercomp_time_avg(procstatus, dscfg)	intercomparison between the average reflectivity of two radars

2.5 Retrievals

process_signal_power(procstatus, dscfg[, ...])	Computes the signal power in dBm

Continued on next page

Table 2.5 – continued from previous page

process_snr(procstatus, dscfg[, radar_list])	Computes SNR
process_l(procstatus, dscfg[, radar_list])	Computes L parameter
process_cdr(procstatus, dscfg[, radar_list])	Computes Circular Depolarization Ratio
process_rainrate(procstatus, dscfg[, radar_list])	Estimates rainfall rate from polarimetric moments
process_wind_vel(procstatus, dscfg[, radar_list])	Estimates the horizontal or vertical component of the wind from the
process_windshear(procstatus, dscfg[, ...])	Estimates the wind shear from the wind velocity

2.6 Trajectory functions

process_trajectory(procstatus, dscfg[, ...])	Return trajectory
process_traj_atplane(procstatus, dscfg[, ...])	Return time series according to trajectory
process_traj_antenna_pattern(procstatus, dscfg)	Process a new array of data volumes considering a plane trajectory.

2.7 COSMO data

process_cosmo(procstatus, dscfg[, radar_list])	Gets COSMO data and put it in radar coordinates
process_cosmo_lookup_table(procstatus, dscfg)	Gets COSMO data and put it in radar coordinates
process_cosmo_coord(procstatus, dscfg[, ...])	Gets the COSMO indices corresponding to each cosmo coordinates
process_hzt(procstatus, dscfg[, radar_list])	Gets iso0 degree data in HZT format and put it in radar coordinates
process_hzt_lookup_table(procstatus, dscfg)	Gets HZT data and put it in radar coordinates
process_hzt_coord(procstatus, dscfg[, ...])	Gets the HZT indices corresponding to each HZT coordinates

pyrad.proc.**get_process_func**(*dataset_type*, *dsname*)

maps the dataset type into its processing function and data set format

Parameters **dataset_type** : str

data set type, i.e. 'RAW', 'SAN', etc.

dsname : str

Name of dataset

Returns **func_name** : str or function

pyrad function used to process the data set type

dsformat : str

data set format, i.e.: 'VOL', etc.

pyrad.proc.**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

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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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_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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**process_correct_noise_rhohv**(*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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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_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. Datset 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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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_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_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]

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_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]

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_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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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_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** : Radar

radar object

ind_rad : int

radar index

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

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. Datset 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** : Radar

radar object

ind_rad : int

radar index

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

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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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. Datset 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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

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

Obtains the radar data at a point measurement

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 of the point of interest

ind_rad : int

radar index

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

Computes quasi vertical profiles

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 50

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_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

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_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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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 20

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.

percent_bins [float. Dataset keyword.] minimum percentage 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 [float. Dataset keyword] maximum standard deviation to consider the data noise. Default 1.

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] multiplicate 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_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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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_time_height**(*procstatus*, *dscfg*, *radar_list=None*)

Produces time height radar objects at a point of interest defined by latitude and longitude

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

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_antenna_pattern**(*procstatus*, *dscfg*, *radar_list=None*, *trajectory=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

data set configuration

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_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

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_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

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.proc.**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_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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

pyrad.proc.**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** : Radar

radar object

ind_rad : int

radar index

CHAPTER

THREE

PRODUCTS GENERATION (PYRAD.PROD)

Initiate the products generation.

3.1 Auxiliary functions

get_dsformat_func	

3.2 Product generation

generate_vol_products(dataset, prdcfg)	generates radar volume products
generate_timeseries_products(dataset, prdcfg)	generates time series products
generate_sun_hits_products(dataset, prdcfg)	generates sun hits products
generate_monitoring_products(dataset, prdcfg)	generates a monitoring product
generate_traj_products	
generate_cosmo_coord_products(dataset, prdcfg)	generates COSMO coordinates products
generate_grid_products(dataset, prdcfg)	generates grid products

pyrad.prod.**generate_cosmo_coord_products**(*dataset*, *prdcfg*)

generates COSMO coordinates 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.**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.**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.**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.**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.**generate_traj_product**(*traj*, *prdcfg*)
Generates trajectory products

Parameters **traj** : Trajectory object

prdcfg : dictionary of dictionaries

product configuration dictionary of dictionaries

Returns None

pyrad.prod.**generate_vol_products**(*dataset*, *prdcfg*)
generates radar volume products

Parameters **dataset** : Radar

radar object

prdcfg : dictionary of dictionaries

product configuration dictionary of dictionaries

Returns no return

pyrad.prod.**get_prodgen_func**(*dsformat*, *dsname*, *dstype*)
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

CHAPTER

FOUR

INPUT AND OUTPUT (pyrad.io)

Functions to read and write data and configuration files.

4.1 Reading configuration files

read_config(fname[, cfg])	Read a pyrad config file.

4.2 Reading radar data

get_data(voltime, datatypesdescr, cfg)	Reads pyrad input data.

4.3 Reading cosmo data

cosmo2radar_data(radar, cosmo_coord, cosmo_data)	get the COSMO value corresponding to each radar gate using nearest
cosmo2radar_coord(radar, cosmo_coord[, ...])	Given the radar coordinates find the nearest COSMO model pixel
hzt2radar_data(radar, hzt_coord, hzt_data[, ...])	get the HZT value corresponding to each radar gate using nearest
hzt2radar_coord(radar, hzt_coord[, ...])	Given the radar coordinates find the nearest HZT pixel
get_cosmo_fields(cosmo_data, cosmo_ind[, ...])	Get the COSMO data corresponding to each radar gate
get_iso0_field(hzt_data, hzt_ind, z_radar[, ...])	Get the height over iso0 data corresponding to each radar gate
read_cosmo_data(fname[, field_names, celsius])	Reads COSMO data from a netcdf file
read_cosmo_coord(fname[, zmin])	Reads COSMO coordinates from a netcdf file
read_hzt_data(fname[, chy0, chx0])	Reads iso-0 degree data from an HZT file

4.4 Reading other data

read_last_state(fname)	Reads a file containing the date of acquisition of the last volume

Continued on next page

Table 4.4 – continued from previous page

read_status(voltime, cfg[, ind_rad])	Reads rad4alp xml status file.
read_rad4alp_cosmo(fname, datatype)	Reads rad4alp COSMO data binary file.
read_rad4alp_vis(fname, datatype)	Reads rad4alp visibility data binary file.
read_colocated_gates(fname)	Reads a csv files containing the posistion of colocated gates
read_colocated_data(fname)	Reads a csv files containing colocated data
read_timeseries(fname)	Reads a time series contained in a csv file
read_ts_cum(fname)	Reads a time series of precipitation accumulation contained in a csv file
read_monitoring_ts(fname)	Reads a monitoring time series contained in a csv file
read_intercomp_scores_ts(fname)	Reads a radar intercomparison scores csv file
get_sensor_data(date, datatype, cfg)	Gets data from a point measurement sensor (rain gauge or disdrometer)
read_smn(fname)	Reads SwissMetNet data contained in a csv file
read_smn2(fname)	Reads SwissMetNet data contained in a csv file with format
read_disdro_scattering(fname)	Reads scattering parameters computed from disdrometer data contained in a
read_sun_hits(fname)	Reads sun hits data contained in a csv file
read_sun_hits_multiple_days(cfg, time_ref[, ...])	Reads sun hits data from multiple file sources
read_sun_retrieval(fname)	Reads sun retrieval data contained in a csv file
read_solar_flux(fname)	Reads solar flux data from the DRAO observatory in Canada
read_selfconsistency(fname)	Reads a self-consistency table with Zdr, Kdp/Zh columns
read_antenna_pattern(fname[, linear, twoway])	Read antenna pattern from file
read_lightning(fname[, filter_data])	Reads lightning data contained in a text file.

4.5 Writing data

send_msg(sender, receiver_list, subject, fname)	sends the content of a text file by email
write_alarm_msg(radar_name, param_name_unit, ...)	writes an alarm file
write_last_state(datetime_last, fname)	writes SwissMetNet data in format datetime,avg_value, std_value
write_smn(datetime_vec, value_avg_vec, ...)	writes SwissMetNet data in format datetime,avg_value, std_value
write_colocated_gates(coloc_gates, fname)	Writes the position of gates colocated with two radars
write_colocated_data(coloc_data, fname)	Writes the data of gates colocated with two radars
write_colocated_data_time_avg(coloc_data, fname)	Writes the time averaged data of gates colocated with two radars
write_timeseries	
write_ts_polar_data(dataset, fname)	writes time series of data
write_ts_cum(dataset, fname)	writes time series accumulation of data
write_monitoring_ts(start_time, np_t, ...)	writes time series of data
write_intercomp_scores_ts(start_time, stats, ...)	writes time series of radar intercomparison scores
write_sun_hits(sun_hits, fname)	Writes sun hits data.
write_sun_retrieval(sun_retrieval, fname)	Writes sun retrieval data.
write_cdf(quantiles, values, ntot, nnan, ...)	writes a cumulative distribution function
write_rhi_profile(hvec, data, nvalid_vec, ...)	writes the values of an RHI profile in a text file
write_field_coverage(quantiles, values, ...)	writes the quantiles of the coverage on a particular sector

4.6 Auxiliary functions

map_hydro(hydro_data_op)	maps the operational hydrometeor classification identifiers to the ones
get_save_dir(basepath, procname, dsname, prdname)	obtains the path to a product directory and eventually creates it
make_filename(prdtype, dstype, dsname, ext)	creates a product file name
get_datetime(fname, datadescriptor)	gets date and time from file name
get_datasetfields	
get_file_list(datadescriptor, starttime, ...)	gets the list of files with a time period
get_datatype_fields(datadescriptor)	splits the data type descriptor and provides each individual member
get_field_unit(datatype)	Return unit of datatype.
get_fieldname_pyart(datatype)	maps the config file radar data type name into the corresponding rainbow
get_fieldname_cosmo(field_name)	maps the Py-ART field name into the corresponding COSMO variable name
generate_field_name_str(datatype)	Generates a field name in a nice to read format.
find_raw_cosmo_file(voltime, datatype, cfg)	Search a COSMO file in netcdf format
find_hzt_file(voltime, cfg[, ind_rad])	Search an ISO-0 degree file in HZT format
add_field(radar_dest, radar_orig)	adds the fields from orig radar into dest radar. If they are not in the
interpol_field(radar_dest, radar_orig, ...)	interpolates field field_name contained in radar_orig to the grid in
get_new_rainbow_file_name(master_fname, ...)	get the rainbow file name containing datatype from a master file name

4.7 Trajectory

Trajectory(filename[, starttime, endtime, ...])	A class for reading and handling trajectory data from a file.

4.8 TimeSeries

TimeSeries(desc[, timevec, timeformat, ...])	Holding timeseries data and metadata.

class pyrad.io.**TimeSeries**(*desc*, *timevec=None*, *timeformat=None*, *maxlength=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

add_dataseries(label, unit_name, unit[, ...])	Add a new data series to the timeseries object.
add_timesample(dt, values)	Add a new sample to the time series.
plot(fname[, ymin, ymax])	Make a figure of a time series
plot_hist(fname[, step])	Make histograms of time series
write(fname)	

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'add_timesample': <function TimeSeries.add_timesample>, 'add_dataseries': <function Ti

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__(*desc*, *timevec=None*, *timeformat=None*, *maxlength=None*, *datatype=''*)
Initalize 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

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyrad.io.timeseries'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

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

__str__
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_dataseries(*label*, *unit_name*, *unit*, *dataseries=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*)

class pyrad.io.**Trajectory**(*filename*, *starttime=None*, *endtime=None*, *trajtype='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 convertion 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

add_radar(radar)	Add the coordinates (WGS84 longitude, latitude and non WGS84 altitude) of a radar to the radar_list.
calculate_velocities(radar)	Calculate velocities.
get_end_time()	Get time of last trajectory sample.
get_samples_in_period([start, end])	"
get_start_time()	Get time of first trajectory sample.

__class__
alias of `type`

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'Trajectory' objects>, '_read_traj_lightning': <

__dir__() → list
default dir() implementation

__eq__
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__gt__
Return self>value.

__hash__
Return hash(self).

__init__ (*filename*, *starttime=None*, *endtime=None*, *trajtype='plane'*, *flashnr=0*)
Initalize 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

__le__
Return self<=value.

__lt__
Return self<value.

__module__ = 'pyrad.io.trajectory'

__ne__
Return self!=value.

__new__()
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
Return repr(self).

__setattr__
Implement setattr(self, name, value).

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

__str__
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

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.

pyrad.io.**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.**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.**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.**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.**find_raw_cosmo_file**(*voltime*, *datatype*, *cfg*, *ind_rad=0*)
Search a COSMO file in netcdf 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

ind_rad : int

radar index

Returns **fname** : str

Name of COSMO file if it exists. None otherwise

pyrad.io.**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.**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.**get_data**(*voltime*, *datatypesdescr*, *cfg*)
Reads pyrad input data.

Parameters **voltime** : 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.**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.**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, 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.**get_datetime**(*fname*, *datadescriptor*)

gets date and time from file name

Parameters **fname** : file name

datadescriptor : str

radar field type. Format : [radar file type]:[datatype]

Returns **fdatetime** : datetime object

date and time in file name

pyrad.io.**get_field_unit**(*datatype*)

Return unit of datatype.

Parameters **datatype** : str

The data type

Returns **unit** : str

The unit

pyrad.io.**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.**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.**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]

startime : 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 **radar** : Radar

radar object

pyrad.io.**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.**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.**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 **savedir** : str

path to product

pyrad.io.**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** : tupple

date, value, type of sensor and measurement period

pyrad.io.**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.**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.**interpol_field**(*radar_dest*, *radar_orig*, *field_name*, *fill_value=None*)

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

Returns **field_dest** : dict

interpolated field and metadata

pyrad.io.**make_filename**(*prdtype*, *dstype*, *dsname*, *ext*, *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 : array 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.**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.**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_colocated_data**(*fname*)
Reads a csv files containing colocated data

Parameters **fname** : str

path of time series file

Returns rad1_ele , rad1_azi, rad1_rng, rad1_val, rad2_ele, rad2_azi, rad2_rng,

rad2_val : tupple

A tupple with the data read. None otherwise

pyrad.io.**read_colocated_gates**(*fname*)
Reads a csv files containing the posistion of colocated gates

Parameters **fname** : str

path of time series file

Returns **rad1_ele , rad1_azi, rad1_rng, rad2_ele, rad2_azi, rad2_rng** : tupple

A tupple with the data read. None otherwise

pyrad.io.**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.**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_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_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 : tupple

The read values

pyrad.io.**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_intercomp_scores_ts**(*fname*)
Reads a radar intercomparison scores csv file

Parameters **fname** : str

path of time series file

Returns date_vec, np_vec, meanbias_vec, medianbias_vec, modebias_vec, corr_vec,

slope_vec, intercep_vec, intercep_slope1_vec : tupple

The read data. None otherwise

pyrad.io.**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_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, time_in_flash, lat, lon, alt, dBm** : tupple

A tupple containing the read values. None otherwise

pyrad.io.**read_monitoring_ts**(*fname*)

Reads a monitoring time series contained in a csv file

Parameters **fname** : str

path of time series file

Returns **date , np_t, central_quantile, low_quantile, high_quantile** : tupple

The read data. None otherwise

pyrad.io.**read_rad4alp_cosmo**(*fname*, *datatype*)

Reads rad4alp COSMO data binary file.

Parameters **fname** : str

name of the file to read

datatype : str

name of the data type

Returns **field** : dictionary

The data field

pyrad.io.**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_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_smn**(*fname*)

Reads SwissMetNet data contained in a csv file

Parameters **fname** : str

path of time series file

Returns **id, date , pressure, temp, rh, precip, wspeed, wdir** : tupple

The read values

pyrad.io.**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 **id, date , value** : tupple

The read values

pyrad.io.**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_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_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, nph, nvalh,

pv, pv_std, npv, nvalv, zdr, zdr_std, nzdr, nvalzdr : tupple

Each parameter is an array containing a time series of information on a variable

pyrad.io.**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 : tupple

Each parameter is an array containing a time series of information on a variable

pyrad.io.**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, nhits_v, el_width_v, az_width_v,

el_bias_v, az_bias_v, dBmv_sun_est, std_dBmv_sun_est, nhits_zdr,

zdr_sun_est, std_zdr_sun_est, dBm_sun_ref, ref_time : tupple

Each parameter is an array containing a time series of information on a variable

pyrad.io.**read_timeseries**(*fname*)

Reads a time series contained in a csv file

Parameters **fname** : str

path of time series file

Returns **date , value** : tupple

A datetime object array containing the time and a numpy masked array containing the value. None otherwise

pyrad.io.**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** : tupple

The data read

pyrad.io.**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_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_cdf**(*quantiles*, *values*, *ntot*, *nnan*, *nclut*, *nblocked*, *nprec_filter*, *noutliers*, *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_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_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_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_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_intercomp_scores_ts**(*start_time*, *stats*, *field_name*, *fname*, *rad1_name='RADAR001'*, *rad2_name='RADAR002'*)

writes time series of radar intercomparison scores

Parameters **start_time** : datetime object

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

Returns **fname** : str

the name of the file where data has written

pyrad.io.**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_monitoring_ts**(*start_time*, *np_t*, *values*, *quantiles*, *datatype*, *fname*)

writes time series of data

Parameters **start_time** : datetime object

the time of the monitoring

np_t : int

the total number of points

values: float array

the values at certain quantiles

quantiles: float array

the quantiles computed

fname : str

file name where to store the data

Returns **fname** : str

the name of the file where data has written

pyrad.io.**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 alitude 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 quantitites 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 specying the sector limits

Returns **fname** : str

the name of the file where data has been written

pyrad.io.**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_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_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_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_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

CHAPTER

FIVE

PLOTTING (PYRAD.GRAPH)

Functions to plot graphics.

5.1 Plots

plot_surface(grid, field_name, level, ...)	plots a surface from gridded data
plot_latitude_slice(grid, field_name, lon, ...)	plots a latitude slice from gridded data
plot_longitude_slice(grid, field_name, lon, ...)	plots a longitude slice from gridded data
plot_latlon_slice(grid, field_name, coord1, ...)	plots a croos section crossing two points in the grid
plot_ppi(radar, field_name, ind_el, prdcfg, ...)	plots a PPI
plot_ppi_map(radar, field_name, ind_el, ...)	plots a PPI on a geographic map
plot_rhi(radar, field_name, ind_az, prdcfg, ...)	plots an RHI
plot_bscope(radar, field_name, ind_sweep, ...)	plots a B-Scope (angle-range representation)
plot_time_range(radar, field_name, ...)	plots a time-range plot
plot_rhi_profile	
plot_along_coord(xval, yval, fname_list[, ...])	plots a time series
plot_field_coverage(xval, yval, fname_list)	plots a time series
plot_density(hist_obj, hist_type, ...[, ...])	density plot (angle-values representation)
plot_cappi(radar, field_name, altitude, ...)	plots a Constant Altitude Plan Position Indicator CAPPI
plot_quantiles(quant, value, fname_list[, ...])	plots quantiles
plot_histogram(bins, values, fname_list[, ...])	computes and plots histogram
plot_histogram2(bins, hist, fname_list[, ...])	plots histogram
plot_antenna_pattern(antpattern, fname_list)	plots an antenna pattern
plot_timeseries(tvec, data, fname_list[, ...])	plots a time series
plot_timeseries_comp(date1, value1, date2, ...)	plots 2 time series in the same graph
plot_monitoring_ts(date, np_t, cquant, ...)	plots a time series of monitoring data
plot_scatter_comp(value1, value2, fname_list)	plots the scatter between two time series
plot_intercomp_scores_ts(date_vec, np_vec, ...)	plots a time series of radar intercomparison scores
plot_sun_hits(field, field_name, fname_list, ...)	plots the sun hits
plot_sun_retrieval_ts(sun_retrieval, ...[, dpi])	plots sun retrieval time series series
get_colobar_label(field_dict, field_name)	creates the colorbar label using field metadata

pyrad.graph.**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.**plot_along_coord**(*xval*, *yval*, *fname_list*, *labelx='coord'*, *labely='Value'*, *labels=None*, *title='Plot along coordinate'*, *colors=None*, *linestyles=None*, *ymin=None*, *ymax=None*, *dpi=72*)

plots a time series

Parameters **xval** : list of float arrays

the x values, range, azimuth or elevation

yval : 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.**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.**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.**plot_cappi**(*radar*, *field_name*, *altitude*, *prdcfg*, *fname_list*)

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

Returns **fname_list** : list of str

list of names of the created plots

pyrad.graph.**plot_density**(*hist_obj, hist_type, field_name, ind_sweep, prdcfg, fname_list, quantiles=[25.0, 50.0, 75.0], ref_value=0.0*)

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

Returns **fname_list** : list of str

list of names of the created plots

pyrad.graph.**plot_field_coverage**(*xval*, *yval*, *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 of float arrays

the x values, azimuth

yval : 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.**plot_histogram**(*bins*, *values*, *fname_list*, *labelx='bins'*, *labely='Number of Samples'*, *titl='histogram'*, *dpi=72*)

computes and plots histogram

Parameters **bins** : array

histogram bins

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.**plot_histogram2**(*bins*, *hist*, *fname_list*, *labelx='bins'*, *labely='Number of Samples'*, *titl='histogram'*, *dpi=72*)

plots histogram

Parameters **quant** : array

histogram bins

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

Returns **fname_list** : list of str

list of names of the created plots

pyrad.graph.**plot_intercomp_scores_ts**(*date_vec*, *np_vec*, *meanbias_vec*, *medianbias_vec*, *modebias_vec*, *corr_vec*, *slope_vec*, *intercep_vec*, *intercep_slope1_vec*, *fname_list*, *ref_value=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

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

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

dpi : int

dots per inch

pyrad.graph.**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.**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 : tupple of floats

lat, lon of the first point

coord2 : tupple 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.**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.**plot_monitoring_ts**(*date*, *np_t*, *cquant*, *lquant*, *hquant*, *field_name*, *fname_list*, *ref_value=None*, *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

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.**plot_ppi**(*radar*, *field_name*, *ind_el*, *prdcfg*, *fname_list*, *plot_type='PPI'*, *step=None*, *quantiles=None*)

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)

step : float

step for histogram plotting

quantiles : float array

quantiles to plot

Returns **fname_list** : list of str

list of names of the created plots

pyrad.graph.**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.**plot_quantiles**(*quant*, *value*, *fname_list*, *labelx='quantile'*, *labely='value'*, *titl='quantile'*, *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

dpi : int

dots per inch

Returns **fname_list** : list of str

list of names of the created plots

pyrad.graph.**plot_rhi**(*radar*, *field_name*, *ind_az*, *prdcfg*, *fname_list*, *plot_type='RHI'*, *step=None*, *quantiles=None*)

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)

step : float

step for histogram plotting

quantiles : float array

quantiles to plot

Returns **fname_list** : list of str

list of names of the created plots

pyrad.graph.**plot_scatter**(*bins1*, *bins2*, *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 **bins1, bins2** : 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 : tupple 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.**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.**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.**plot_sun_retrieval_ts**(*sun_retrieval*, *data_type*, *fname_list*, *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

dpi : int

dots per inch

Returns **fname_list** : list of str

list of names of the created plots

pyrad.graph.**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_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.**plot_timeseries**(*tvec, data, fname_list, labelx='Time [UTC]', labely='Value', labels=['Sensor'], title='Time Series', period=0, timeformat=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 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

labely : 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.**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*, *period2=0*, *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

Returns **fname_list** : list of str

list of names of the created plots

CHAPTER

SIX

UTILITIES (PYRAD.UTIL)

Functions to read and write data and configuration files.

6.1 Radar Utilities

get_ROI(radar, fieldname, sector)	filter out any data outside the region of interest defined by sector
rainfall_accumulation(t_in_vec, val_in_vec)	Computes the rainfall accumulation of a time series over a given period
time_series_statistics(t_in_vec, val_in_vec)	Computes statistics over a time-averaged series
join_time_series(t1, val1, t2, val2[, dropnan])	joins time_series
get_range_bins_to_avg(rad1_rng, rad2_rng)	Compares the resolution of two radars and determines if and which radar
find_ray_index(ele_vec, azi_vec, ele, azi[, ...])	Find the ray index corresponding to a particular elevation and azimuth
find_rng_index(rng_vec, rng[, rng_tol])	Find the range index corresponding to a particular range
time_avg_range(timeinfo, avg_starttime, ...)	finds the new start and end time of an averaging
get_closest_solar_flux(hit_datetime_list, ...)	finds the solar flux measurement closest to the sun hit
create_sun_hits_field(rad_el, rad_az, ...)	creates a sun hits field from the position and power of the sun hits
create_sun_retrieval_field(par, imgcfg)	creates a sun retrieval field from the retrieval parameters
compute_quantiles(field[, quantiles])	computes quantiles
compute_quantiles_from_hist(bins, hist[, ...])	computes quantiles from histograms
compute_quantiles_sweep(field, ray_start, ...)	computes quantiles of a particular sweep
compute_2d_hist(field1, field2, field_name1, ...)	computes histogram of the data
compute_1d_stats(field1, field2)	returns statistics of data
compute_2d_stats(field1, field2, ...[, ...])	computes a 2D histogram and statistics of the data
compute_histogram(field, field_name[, step])	computes histogram of the data
compute_histogram_sweep(field, ray_start, ...)	computes histogram of the data in a particular sweep
quantiles_weighted(values[, weight_vector, ...])	Given a set of values and weights, compute the weighted quantile(s).

pyrad.util.**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.**compute_2d_hist**(*field1*, *field2*, *field_name1*, *field_name2*, *step1=None*, *step2=None*)
computes histogram of the data

Parameters **field** : ndarray 2D

the radar field

field_name: str

name of the field

step : float

size of bin

Returns **bins** : float array

interval of each bin

values : float array

values at each bin

pyrad.util.**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_nam2: str

the name of the fields

step1, step2 : float

size of bin

Returns **hist_2d** : array

the histogram

bins1, bins2 : float array

interval of each bin

stats : dict

a dictionary with statistics

pyrad.util.**compute_histogram**(*field*, *field_name*, *step=None*)
computes histogram of the data

Parameters **field** : ndarray 2D

the radar field

field_name: str

name of the field

step : float

size of bin

Returns **bins** : float array

interval of each bin

values : float array

values at each bin

pyrad.util.**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 **bins** : float array

interval of each bin

values : float array

values at each bin

pyrad.util.**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.**compute_quantiles_from_hist**(*bins*, *hist*, *quantiles=None*)
computes quantiles from histograms

Parameters **bins** : 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.**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.**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.**create_sun_retrieval_field**(*par*, *imgcfg*)
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.**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.**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.**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.**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.**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.**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.**quantiles_weighted**(*values*, *weight_vector=None*, *quantiles=array([0.5])*, *weight_threshold=None*, *data_is_log=False*)

Given a set of values and weights, compute the weighted quantile(s).

pyrad.util.**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.**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.**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

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

p

Symbols

A

C

F

G

H

I

J

M

P

Q

R

S

T

W