
pyrad library reference for developers

Release 0.0.1

meteoswiss-mdr

Oct 20, 2016

CONTENTS

1	pyrad.flow.flow_control	3
2	pyrad.proc.process_aux	7
3	pyrad.proc.process_echoclass	9
4	pyrad.proc.process_phase	11
5	pyrad.proc.process_retrieve	17
6	pyrad.proc.process_calib	21
7	pyrad.prod.product_aux	25
8	pyrad.prod.process_product	27
9	pyrad.io.read_data_radar	29
10	pyrad.io.read_data_other	33
11	pyrad.io.write_data	37
12	pyrad.io.io_aux	39
13	pyrad.graph.plots	43
14	pyrad.util.radar_utils	49
15	Indices and tables	51
	Python Module Index	53
	Index	55

Contents:

PYRAD.FLOW.FLOW_CONTROL

functions to control the Pyrad data processing flow

<code>main(cfgfile, starttime, endtime)</code>	main flow control. Processes data over a given period of time
<code>_create_cfg_dict(cfgfile)</code>	creates a configuration dictionary
<code>_create_datacfg_dict(cfg)</code>	creates a data configuration dictionary from a config dictionary
<code>_create_dscfg_dict(cfg, dataset[, voltime])</code>	creates a dataset configuration dictionary
<code>_create_prdcfg_dict(cfg, dataset, product[, ...])</code>	creates a product configuration dictionary
<code>_get_datatype_list(cfg)</code>	get list of unique input data types
<code>_get_datasets_list(cfg)</code>	get list of dataset at each processing level
<code>_get_masterfile_list(masterscan, ...)</code>	get master file list
<code>_add_dataset(new_dataset, radar[, make_global])</code>	adds a new field to an existing radar object
<code>_process_dataset(cfg, dataset[, ...])</code>	processes a dataset

`pyrad.flow.flow_control._add_dataset (new_dataset, radar, make_global=True)`
adds a new field to an existing radar object

Parameters `new_dataset` : radar object

the radar object containing the new fields

radar : radar object

the radar object containing the global data

make_global : boolean

if true a new field is added to the global data

Returns 0 if successful. None otherwise

`pyrad.flow.flow_control._create_cfg_dict (cfgfile)`
creates a configuration dictionary

Parameters `cfgfile` : str

path of the main config file

Returns `cfg` : dict

dictionary containing the configuration data

`pyrad.flow.flow_control._create_datacfg_dict (cfg)`
creates a data configuration dictionary from a config dictionary

Parameters `cfg` : dict

config dictionary

Returns datacfg : dict

data config dictionary

`pyrad.flow.flow_control._create_dscfg_dict (cfg, dataset, voltime=None)`
creates a dataset configuration dictionary

Parameters cfg : dict

config dictionary

dataset : str

name of the dataset

voltime : datetime object

time of the dataset

Returns dscfg : dict

dataset config dictionary

`pyrad.flow.flow_control._create_prdcfg_dict (cfg, dataset, product, voltime=None)`
creates a product configuration dictionary

Parameters cfg : dict

config dictionary

dataset : str

name of the dataset used to create the product

product : str

name of the product

voltime : datetime object

time of the dataset

Returns prdcfg : dict

product config dictionary

`pyrad.flow.flow_control._get_datasets_list (cfg)`
get list of dataset at each processing level

Parameters cfg : dict

config dictionary

Returns dataset_levels : dict

a dictionary containing the list of datasets at each processing level

`pyrad.flow.flow_control._get_datatype_list (cfg)`
get list of unique input data types

Parameters cfg : dict

config dictionary

Returns datatypesdescr : list

list of data type descriptors

`pyrad.flow.flow_control._get_masterfile_list` (*masterscan, datatypesdescr, starttime, endtime, datacfg*)

get master file list

Parameters `masterscan` : str

name of the master scan

datatypesdescr : list

list of unique data type descriptors

starttime, endtime : datetime object

start and end of processing period

datacfg : dict

data configuration dictionary

Returns `masterfilelist` : list

the list of master files

masterdatatypesdescr : str

the master data type descriptor

`pyrad.flow.flow_control._process_dataset` (*cfg, dataset, proc_status=0, radar=None, voltime=None*)

processes a dataset

Parameters `cfg` : dict

configuration dictionary

dataset : str

name of the dataset to be processed

proc_status : int

status of the processing 0: Initialization 1: process of radar volume 2: Final processing

radar : radar object

radar object containing the data to be processed

voltime : datetime object

reference time of the radar

Returns 0 if a new dataset has been created. None otherwise

`pyrad.flow.flow_control.main` (*cfgfile, starttime, endtime*)

main flow control. Processes data over a given period of time

Parameters `cfgfile` : str

path of the main config file

starttime, endtime : datetime object

start and end time of the data to be processed

Returns None

PYRAD.PROC.PROCESS_AUX

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

<code>get_process_type(dataset_type)</code>	maps the dataset type into its processing function and data set format
<code>process_raw(procstatus, dscfg[, radar])</code>	dummy function that returns the initial input data set
<code>process_save_radar(procstatus, dscfg[, radar])</code>	dummy function that allows to save the entire radar object
<code>process_point_measurement(procstatus, dscfg)</code>	Obtains the radar data at a point measurement

`pyrad.proc.process_aux.get_process_type(dataset_type)`
maps the dataset type into its processing function and data set format

Parameters `dataset_type` : str

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

Returns `func_name` : str

pyrad function used to process the data set type

dsformat : str

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

`pyrad.proc.process_aux.process_point_measurement(procstatus, dscfg, radar=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 : Radar

Optional. Radar object

Returns new_dataset : dict

dictionary containing the data and metadata of the point of interest

`pyrad.proc.process_aux.process_raw(procstatus, dscfg, radar=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 : Radar

Optional. Radar object

Returns new_dataset : Radar

radar object

`pyrad.proc.process_aux.process_save_radar(procstatus, dscfg, radar=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 : Radar

Optional. Radar object

Returns new_dataset : Radar

radar object

PYRAD.PROC.PROCESS_ECHOCCLASS

Functions for echo classification and filtering

<code><i>process_echo_id</i>(procstatus, dscfg[, radar])</code>	identifies echoes as 0: No data, 1: Noise, 2: Clutter,
<code><i>process_echo_filter</i>(procstatus, dscfg[, radar])</code>	Masks all echo types that are not of the class specified in
<code><i>process_filter_snr</i>(procstatus, dscfg[, radar])</code>	filters out low SNR echoes
<code><i>process_hydroclass</i>(procstatus, dscfg[, radar])</code>	Classifies precipitation echoes

`pyrad.proc.process_echoclass.process_echo_filter (procstatus, dscfg, radar=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

radar : Radar

Optional. Radar object

Returns `new_dataset` : Radar

radar object

`pyrad.proc.process_echoclass.process_echo_id (procstatus, dscfg, radar=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 : Radar

Optional. Radar object

Returns `new_dataset` : Radar

radar object

`pyrad.proc.process_echoclass.process_filter_snr(procstatus, dscfg, radar=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 : Radar

Optional. Radar object

Returns `new_dataset` : Radar

radar object

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

Parameters `procstatus` : int

Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg : dictionary of dictionaries

data set configuration. Accepted Configuration Keywords:

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

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

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

radar : Radar

Optional. Radar object

Returns `radar` : Radar

radar object

PYRAD.PROC.PROCESS_PHASE

Functions for PhiDP and KDP processing and attenuation correction

<code>process_estimate_phidp0(procstatus, dscfg[, ...])</code>	estimates the system differential phase offset at each ray
<code>process_correct_phidp0(procstatus, dscfg[, ...])</code>	corrects phidp of the system phase
<code>process_smooth_phidp_single_window(...[, radar])</code>	corrects phidp of the system phase and smoothes it using one window
<code>process_smooth_phidp_double_window(...[, radar])</code>	corrects phidp of the system phase and smoothes it using one window
<code>process_kdp_leastsquare_single_window(...[, ...])</code>	Computes specific differential phase using a piecewise least square method
<code>process_kdp_leastsquare_double_window(...[, ...])</code>	Computes specific differential phase using a piecewise least square method
<code>process_phidp_kdp_Maesaka(procstatus, dscfg)</code>	Estimates PhiDP and KDP using the method by Maesaka
<code>process_phidp_kdp_lp(procstatus, dscfg[, radar])</code>	Estimates PhiDP and KDP using a linear programming algorithm
<code>process_selfconsistency_kdp_phidp</code>	
<code>process_selfconsistency_bias</code>	
<code>process_attenuation(procstatus, dscfg[, radar])</code>	Computes specific attenuation and specific differential attenuation using

`pyrad.proc.process_phase.process_attenuation` (*procstatus*, *dscfg*, *radar=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

radar : Radar

Optional. Radar object

Returns `radar` : Radar

radar object

`pyrad.proc.process_phase.process_correct_phidp0` (*procstatus, dscfg, radar=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` : Radar

Optional. Radar object

Returns `new_dataset` : Radar

radar object

`pyrad.proc.process_phase.process_estimate_phidp0` (*procstatus, dscfg, radar=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` : Radar

Optional. Radar object

Returns `new_dataset` : Radar

radar object

`pyrad.proc.process_phase.process_kdp_leastsquare_double_window` (*procstatus,*
dscfg,
radar=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 : Radar

Optional. Radar object

Returns radar : Radar

radar object

`pyrad.proc.process_phase.process_kdp_leastsquare_single_window` (*procstatus*,
dscfg,
radar=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 : Radar

Optional. Radar object

Returns radar : Radar

radar object

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

Estimates PhiDP and KDP using the method by Maesaka

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 : Radar

Optional. Radar object

Returns **new_dataset** : Radar

radar object

`pyrad.proc.process_phase.process_phidp_kdp_lp(procstatus, dscfg, radar=None)`
Estimates PhiDP and KDP using a linear programming algorithm

Parameters **procstatus** : int

Processing status: 0 initializing, 1 processing volume, 2 post-processing

dscfg : dictionary of dictionaries

data set configuration. Accepted Configuration Keywords:

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

radar : Radar

Optional. Radar object

Returns **new_dataset** : Radar

radar object

`pyrad.proc.process_phase.process_smooth_phidp_double_window(procstatus, dscfg, radar=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 : Radar

Optional. Radar object

Returns `new_dataset` : Radar

radar object

`pyrad.proc.process_phase.process_smooth_phidp_single_window` (*procstatus*, *dscfg*,
radar=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 : Radar

Optional. Radar object

Returns `new_dataset` : Radar

radar object

PYRAD.PROC.PROCESS_RETRIEVE

Functions for retrieving new moments and products

<code>process_signal_power(procstatus, dscfg[, radar])</code>	Computes the signal power in dBm
<code>process_snr(procstatus, dscfg[, radar])</code>	Computes SNR
<code>process_l(procstatus, dscfg[, radar])</code>	Computes L parameter
<code>process_cdr(procstatus, dscfg[, radar])</code>	Computes Circular Depolarization Ratio
<code>process_rainrate(procstatus, dscfg[, radar])</code>	Estimates rainfall rate from polarimetric moments

`pyrad.proc.process_retrieve.process_cdr (procstatus, dscfg, radar=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 : Radar

Optional. Radar object

Returns `new_dataset` : Radar

radar object

`pyrad.proc.process_retrieve.process_l (procstatus, dscfg, radar=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 : Radar

Optional. Radar object

Returns `new_dataset` : Radar

radar object

`pyrad.proc.process_retrieve.process_rainrate` (*procstatus*, *dscfg*, *radar=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 : Radar

Optional. Radar object

Returns *radar* : Radar

radar object

`pyrad.proc.process_retrieve.process_signal_power` (*procstatus*, *dscfg*, *radar=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

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

attg [float. Dataset keyword] The gas attenuation

radar : Radar

Optional. Radar object

Returns *new_dataset* : Radar

radar object

`pyrad.proc.process_retrieve.process_snr` (*procstatus*, *dscfg*, *radar=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 : Radar

Optional. Radar object

Returns **new_dataset** : Radar

radar object

PYRAD.PROC.PROCESS_CALIB

Functions for monitoring data quality and correct bias and noise effects

<code>process_correct_bias(procstatus, dscfg[, radar])</code>	Corrects a bias on the data
<code>process_correct_noise_rhohv(procstatus, dscfg)</code>	identifies echoes as 0: No data, 1: Noise, 2: Clutter,
<code>process_selfconsistency_kdp_phidp(...[, radar])</code>	Computes specific differential phase and differential phase in rain using
<code>process_selfconsistency_bias(procstatus, dscfg)</code>	Estimates the reflectivity bias by means of the selfconsistency
<code>process_rhohv_rain(procstatus, dscfg[, radar])</code>	Computes quantiles of RhoHV in rain
<code>process_sun_hits(procstatus, dscfg[, radar])</code>	monitoring of the radar using sun hits

`pyrad.proc.process_calib.process_correct_bias` (*procstatus*, *dscfg*, *radar=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]

radar : Radar

Optional. Radar object

Returns *new_dataset* : Radar

radar object

`pyrad.proc.process_calib.process_correct_noise_rhohv` (*procstatus*, *dscfg*,
radar=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 : Radar

Optional. Radar object

Returns new_dataset : Radar

radar object

`pyrad.proc.process_calib.process_rhohv_rain(procstatus, dscfg, radar=None)`
 Computes quantiles 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]

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

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

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

radar : Radar

Optional. Radar object

Returns radar : Radar

radar object

`pyrad.proc.process_calib.process_selfconsistency_bias(procstatus, dscfg, radar=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

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

radar : Radar

Optional. Radar object

Returns radar : Radar

radar object

`pyrad.proc.process_calib.process_selfconsistency_kdp_phidp(procstatus, dscfg, radar=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]

radar : Radar

Optional. Radar object

Returns radar : Radar

radar object

`pyrad.proc.process_calib.process_sun_hits` (*procstatus, dscfg, radar=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]

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

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

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

percent_bins [float. Dataset keyword] minimum percentage of range bins that have to contain signal to consider the ray a potential sun hit

attg [float. Dataset keyword] gaseous attenuation

max_std [float. Dataset keyword] maximum standard deviation to consider the data noise

az_width_co [float. Dataset keyword] co-polar antenna azimuth width (convoluted with sun width) [deg]

el_width_co [float. Dataset keyword] co-polar antenna elevation width (convoluted with sun width) [deg]

az_width_cross [float. Dataset keyword] cross-polar antenna azimuth width (convoluted with sun width) [deg]

el_width_cross [float. Dataset keyword] cross-polar antenna elevation width (convoluted with sun width) [deg]

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

radar : Radar

Optional. Radar object

Returns `sun_hits_dict` : dict

dictionary containing a radar object, a `sun_hits` dict and a `sun_retrieval` dictionary

PYRAD.PROD.PRODUCT_AUX

Auxiliary functions to generate products

<code>get_product_type</code> (product_type)	maps the product type into its processing function
--	--

`pyrad.prod.product_aux.get_product_type` (*product_type*)

maps the product type into its processing function

Parameters `product_type` : str

product type, i.e. 'VOL', etc.

Returns `func_name` : str

pyrad function used to generate the product

PYRAD.PROD.PROCESS_PRODUCT

Functions for obtaining Pyrad products from the datasets

<code>get_product_type</code>	
<code>generate_vol_products(dataset, prdcfg)</code>	generates radar volume products
<code>generate_sun_hits_products(dataset, prdcfg)</code>	generates sun hits products
<code>generate_timeseries_products(dataset, prdcfg)</code>	generates time series products
<code>get_save_dir(basepath, procname, dsname, prdname)</code>	obtains the path to a product directory and eventually creates it
<code>make_filename(prdtype, dstype, dsname, ext)</code>	creates a product file name

`pyrad.prod.process_product.generate_sun_hits_products(dataset, prdcfg)`
generates sun hits products

Parameters dataset : tuple

radar object and sun hits dictionary

prdcfg : dictionary of dictionaries

product configuration dictionary of dictionaries

Returns filename : str

the name of the file created. None otherwise

`pyrad.prod.process_product.generate_timeseries_products(dataset, prdcfg)`
generates time series products

Parameters dataset : dictionary

radar object

prdcfg : dictionary of dictionaries

product configuration dictionary of dictionaries

Returns no return

`pyrad.prod.process_product.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.IO.READ_DATA_RADAR

Functions for reading radar data files

<i>get_data</i> (voltime, datatypesdescr, cfg)	Reads pyrad input data.
<i>merge_scans_rainbow</i> (basepath, scan_list, ...)	merge rainbow scans
<i>merge_scans_rad4alp</i> (basepath, scan_list, ...)	merge rad4alp data.
<i>merge_scans_cosmo</i> (voltime, datatype_list, cfg)	merge rainbow scans
<i>merge_scans_cosmo_rad4alp</i> (voltime, datatype, cfg)	merge cosmo rad4alp scans. If data for all the scans cannot be retrieved
<i>merge_fields_rainbow</i> (basepath, scan_name, ...)	merge Rainbow fields into a single radar object.
<i>merge_fields_cosmo</i> (filename_list)	merge COSMO fields in Rainbow file format
<i>get_data_rainbow</i> (filename, datatype)	gets rainbow radar data
<i>get_data_rad4alp</i> (filename, datatype_list, ...)	gets rad4alp radar data

`pyrad.io.read_data_radar.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.read_data_radar.get_data_rad4alp` (*filename*, *datatype_list*, *scan_name*, *cfg*)

gets rad4alp radar data

Parameters **filename** : str

name of file containing rainbow data

datatype_list : list of strings

list of data fields to get

scan_name : str

name of the elevation (001 to 020)

cfg : dict

configuration dictionary

Returns radar : Radar

radar object

`pyrad.io.read_data_radar.get_data_rainbow(filename, datatype)`

gets rainbow radar data

Parameters filename : str

name of file containing rainbow data

datatype : str

field name

Returns radar : Radar

radar object

`pyrad.io.read_data_radar.merge_fields_cosmo(filename_list)`

merge COSMO fields in Rainbow file format

Parameters filename_list : str

list of file paths where to find the data

Returns radar : Radar

radar object

`pyrad.io.read_data_radar.merge_fields_rainbow(basepath, scan_name, voltime, datatype_list)`

merge Rainbow fields into a single radar object.

Parameters basepath : str

name of the base path where to find the data

scan_name: str

name of the scan

voltime : datetime object

reference time of the scan

datatype_list : list

lists of data types to get

Returns radar : Radar

radar object

`pyrad.io.read_data_radar.merge_scans_cosmo(voltime, datatype_list, cfg)`

merge rainbow scans

Parameters voltime: datetime object

reference time of the scan

datatype_list : list

lists of data types to get

cfg : dict

configuration dictionary

Returns radar : Radar

radar object

`pyrad.io.read_data_radar.merge_scans_cosmo_rad4alp` (*voltime, datatype, cfg*)
merge cosmo rad4alp scans. If data for all the scans cannot be retrieved returns None

Parameters voltime: datetime object

reference time of the scan

datatype_list : list

lists of data types to get

cfg : dict

configuration dictionary

Returns radar : Radar

radar object

`pyrad.io.read_data_radar.merge_scans_rad4alp` (*basepath, scan_list, radar_name, radar_res, voltime, datatype_list, cfg*)
merge rad4alp data.

Parameters basepath : str

base path of rad4alp radar data

scan_list : list

list of scans (001 to 020)

radar_name : str

radar_name (A, D, L, ...)

radar_res : str

radar resolution (H or L)

voltime: datetime object

reference time of the scan

datatype_list : list

lists of data types to get

cfg : dict

configuration dictionary

Returns radar : Radar

radar object

`pyrad.io.read_data_radar.merge_scans_rainbow` (*basepath, scan_list, voltime, scan_period, datatype_list, cfg*)
merge rainbow scans

Parameters basepath : str

base path of rad4alp radar data

scan_list : list

list of scans

voltime: datetime object

reference time of the scan

scan_period : float

time from reference time where to look for other scans data

datatype_list : list

lists of data types to get

cfg : dict

configuration dictionary

Returns radar : Radar

radar object

PYRAD.IO.READ_DATA_OTHER

Functions for reading auxiliary data

<code>read_status(voltime, cfg)</code>	Reads rad4alp xml status file.
<code>read_rad4alp_cosmo(fname, datatype)</code>	Reads rad4alp COSMO data binary file.
<code>read_timeseries(fname)</code>	Reads a time series contained in a csv file
<code>read_sun_hits_multiple_days(cfg[, nfiles])</code>	Reads sun hits data from multiple file sources
<code>read_sun_hits(fname)</code>	Reads sun hits data contained in a csv file
<code>read_sun_retrieval(fname)</code>	Reads sun retrieval data contained in a csv file
<code>get_sensor_data(date, datatype, cfg)</code>	Gets data from a point measurement sensor (rain gauge or disdrometer)
<code>read_smn(fname)</code>	Reads SwissMetNet data contained in a csv file
<code>read_disdro_scattering(fname)</code>	Reads scattering parameters computed from disdrometer data contained in a
<code>read_selfconsistency(fname)</code>	Reads a self-consistency table with Zdr, Kdp/Zh columns

`pyrad.io.read_data_other.get_sensor_data (date, datatype, cfg)`

Gets data from a point measurement sensor (rain gauge or disdrometer)

Parameters `date` : datetime object

measurement date

datatype : str

name of the data type to read

cfg : dictionary

dictionary containing sensor information

Returns `sensordate , sensorvalue, label, period` : tuple

date, value, type of sensor and measurement period

`pyrad.io.read_data_other.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 `id, date , pressure, temp, rh, precip, wspeed, wdir` : arrays

The read values

`pyrad.io.read_data_other.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_data_other.read_selfconsistency(fname)`

Reads a self-consistency table with Zdr, Kdp/Zh columns

Parameters **fname** : str

path of time series file

Returns **zdr, kdpzh** : arrays

The read values

`pyrad.io.read_data_other.read_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** : tuple

The read values

`pyrad.io.read_data_other.read_status(voltime, cfg)`

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

Returns **root** : root element object

The information contained in the status file

`pyrad.io.read_data_other.read_sun_hits(fname)`

Reads sun hits data contained in a csv file

Parameters **fname** : str

path of time series file

Returns **date, ray, nrng, rad_el, rad_az, sun_el, sun_az, ph, ph_std, npv, nvalh,**

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

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

`pyrad.io.read_data_other.read_sun_hits_multiple_days(cfg, nfiles=1)`

Reads sun hits data from multiple file sources

Parameters **cfg** : dict

dictionary with configuration data to find out the right file

nfiles : int

number of files to read

Returns date, ray, nrng, rad_el, rad_az, sun_el, sun_az, ph, ph_std, nph, nvalh,

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

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

`pyrad.io.read_data_other.read_sun_retrieval(fname)`

Reads sun retrieval data contained in a csv file

Parameters **fname** : str

path of time series file

Returns 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 : tuple

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

`pyrad.io.read_data_other.read_timeseries(fname)`

Reads a time series contained in a csv file

Parameters **fname** : str

path of time series file

Returns **date, value** : tuple

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

PYRAD.IO.WRITE_DATA

Functions for writing pyrad output data

<code>write_timeseries(dataset, fname)</code>	writes time series of data
<code>write_sun_hits(sun_hits, fname)</code>	Writes sun hits data.
<code>write_sun_retrieval(sun_retrieval, fname)</code>	Writes sun retrieval data.
<code>generate_field_name_str(datatype)</code>	Generates a field name in a nice to read format.

`pyrad.io.write_data.write_sun_hits(sun_hits, fname)`
Writes sun hits data.

Parameters `sun_hits` : dict
dictionary containing the sun hits parameters
`fname` : str
file name where to store the data

Returns `fname` : str
the name of the file where data has written

`pyrad.io.write_data.write_sun_retrieval(sun_retrieval, fname)`
Writes sun retrieval data.

Parameters `sun_retrieval` : dict
dictionary containing the sun retrieval parameters
`fname` : str
file name where to store the data

Returns `fname` : str
the name of the file where data has written

`pyrad.io.write_data.write_timeseries(dataset, fname)`
writes time series of data

Parameters `dataset` : dict
dictionary containing the time series parameters
`fname` : str
file name where to store the data

Returns `fname` : str

the name of the file where data has written

PYRAD.IO.IO_AUX

Auxiliary functions for reading/writing files

<code>get_save_dir(basepath, procname, dsname, prdname)</code>	obtains the path to a product directory and eventually creates it
<code>make_filename(prdtype, dstype, dsname, ext)</code>	creates a product file name
<code>generate_field_name_str(datatype)</code>	Generates a field name in a nice to read format.
<code>get_datatype_metranet(datatype)</code>	maps de config file radar data type name into the corresponding metranet
<code>get_fieldname_pyart(datatype)</code>	maps de config file radar data type name into the corresponding rainbow
<code>get_file_list(scan, datadescriptor, ...)</code>	gets the list of files with a time period
<code>get_datatype_fields(datadescriptor)</code>	splits the data type descriptor and provides each individual member
<code>get_dataset_fields(datasetdescr)</code>	splits the dataset type descriptor and provides each individual member
<code>get_datetime(fname, datadescriptor)</code>	gets date and time from file name
<code>find_cosmo_file(voltime, datatype, cfg, scanid)</code>	Search a COSMO file
<code>find_rad4alpcosmo_file(voltime, datatype, ...)</code>	Search a COSMO file

`pyrad.io.io_aux.find_cosmo_file (voltime, datatype, cfg, scanid)`
Search a COSMO file

Parameters **voltime** : datetime object

volume scan time

datatype : type of COSMO data to look for

cfg: dictionary of dictionaries

configuration info to figure out where the data is

scanid: str

name of the scan

Returns **fname** : str

Name of COSMO file if it exists. None otherwise

`pyrad.io.io_aux.find_rad4alpcosmo_file (voltime, datatype, cfg, scanid)`
Search a COSMO file

Parameters **voltime** : datetime object

volume scan time

datatype : type of COSMO data to look for

cfg: dictionary of dictionaries

configuration info to figure out where the data is

Returns fname : str

Name of COSMO file if it exists. None otherwise

scanid: str

name of the scan

`pyrad.io.io_aux.generate_field_name_str (datatype)`

Generates a field name in a nice to read format.

Parameters datatype : str

The data type

Returns field_str : str

The field name

`pyrad.io.io_aux.get_dataset_fields (datasetdescr)`

splits the dataset type descriptor and provides each individual member

Parameters datasetdescr : str

dataset type. Format : [processing level]:[dataset type]

Returns proclevel : str

dataset processing level

dataset : str

dataset type, i.e. dBZ, ZDR, ISO0, ...

`pyrad.io.io_aux.get_datatype_fields (datadescriptor)`

splits the data type descriptor and provides each individual member

Parameters datadescriptor : str

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

Returns datagroup : str

data type group, i.e. RAINBOW, RAD4ALP, SAVED, COSMO, ...

datatype : str

data type, i.e. dBZ, ZDR, ISO0, ...

dataset : str

dataset type (for saved data only)

product : str

product type (for saved data only)

`pyrad.io.io_aux.get_datatype_metrane (datatype)`

maps de config file radar data type name into the corresponding metranet data type name and Py-ART field name

Parameters datatype : str

config file radar data type name

Returns metranet type : dict

dictionary containing the metranet data type name and its corresponding Py-ART field name

`pyrad.io.io_aux.get_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.io_aux.get_fieldname_pyart(datatype)`

maps de config file radar data type name into the corresponding rainbow Py-ART field name

Parameters datatype : str

config file radar data type name

Returns field_name : str

Py-ART field name

`pyrad.io.io_aux.get_file_list(scan, datadescriptor, starttime, endtime, cfg)`

gets the list of files with a time period

Parameters scan : str

scan name

datadescriptor : str

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

starttime : datetime object

start of time period

endtime : datetime object

end of time period

cfg: dictionary of dictionaries

configuration info to figure out where the data is

Returns radar : Radar

radar object

`pyrad.io.io_aux.get_save_dir(basepath, procname, dsname, prdname, timeinfo=None, timeformat='%Y-%m-%d', create_dir=True)`

obtains the path to a product directory and eventually creates it

Parameters basepath : str

product base path

procname : str

name of processing space

dsname : str

data set name

prdname : str

product name

timeinfo : datetime

time info to generate the date directory. If None there is no time format in the path

timeformat : str

Optional. The time format.

create_dir : boolean

If True creates the directory

Returns savedir : str

path to product

`pyrad.io.io_aux.make_filename` (*prdtype, dstype, dsname, ext, prdcfginfo=None, timeinfo=None, timeformat='%Y%m%d%H%M%S'*)

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 : str

file name extension, i.e. 'png'

prdcfginfo : str

Optional. string to add product configuration information, i.e. 'el0.4'

timeformat : str

Optional. The time format

Returns fname : str

file name

PYRAD.GRAPH.PLOTS

Functions to plot Pyrad datasets

<code>plot_ppi(radar, field_name, ind_el, prdcfg, ...)</code>	plots a PPI
<code>plot_rhi(radar, field_name, ind_az, prdcfg, ...)</code>	plots an RHI
<code>plot_cappi(radar, field_name, altitude, ...)</code>	plots a Constant Altitude Plan Position Indicator CAPPI
<code>plot_bscope(radar, field_name, ind_sweep, ...)</code>	plots a B-Scope (angle-range representation)
<code>plot_sun_hits(field, field_name, fname, prdcfg)</code>	plots the sun hits
<code>plot_quantiles(quant, value, fname[, ...])</code>	plots quantiles
<code>plot_timeseries(date, value, fname[, ...])</code>	plots a time series
<code>plot_timeseries_comp(date1, value1, date2, ...)</code>	plots 2 time series in the same graph
<code>plot_sun_retrieval_ts(sun_retrieval, ...)</code>	plots a time series
<code>get_colobar_label(field_dict, field_name)</code>	creates the colorbar label using field metadata
<code>get_field_name(field_dict, field)</code>	Return a nice field name for a particular field

`pyrad.graph.plots.get_colobar_label` (*field_dict*, *field_name*)
creates the colorbar label using field metadata

Parameters **field_dict** : dict
dictionary containing field metadata
field_name : str
name of the field

Returns **label** : str
colorbar label

`pyrad.graph.plots.get_field_name` (*field_dict*, *field*)
Return a nice field name for a particular field

Parameters **field_dict** : dict
dictionary containing field metadata
field : str
name of the field

Returns **field_name** : str
the field name

`pyrad.graph.plots.plot_bscope` (*radar*, *field_name*, *ind_sweep*, *prdcfg*, *fname*)
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 : str

name of the file where to store the plot

Returns **fname** : str

the name of the created plot file

`pyrad.graph.plots.plot_cappi (radar, field_name, altitude, prdcfg, fname)`
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 : str

name of the file where to store the plot

Returns **fname** : str

the name of the created plot file

`pyrad.graph.plots.plot_histogram (bins, values, fname, labelx='bins', labely='Number of Samples', titl='histogram')`

plots histogram

Parameters **quant** : array

quantiles to be plotted

value : array

values of each quantie

fname : str

name of the file where to store the plot

labelx : str

The label of the X axis

lably : str

The label of the Y axis

titl : str

The figure title

Returns fname : str

the name of the created plot file

`pyrad.graph.plots.plot_ppi (radar, field_name, ind_el, prdcfg, fname, 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 : str

name of the file 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 : str

the name of the created plot file

`pyrad.graph.plots.plot_quantiles (quant, value, fname, labelx='quantile', labely='value', titl='quantile')`

plots quantiles

Parameters quant : array

quantiles to be plotted

value : array

values of each quantie

fname : str

name of the file where to store the plot

labelx : str

The label of the X axis

labeled : str

The label of the Y axis

title : str

The figure title

Returns filename : str

the name of the created plot file

`pyrad.graph.plots.plot_rhi(radar, field_name, ind_az, prdcfg, fname, plot_type='PPI',
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 : str

name of the file 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 filename : str

the name of the created plot file

`pyrad.graph.plots.plot_sun_hits(field, field_name, fname, 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 : str

name of the file where to store the plot

Returns fname : str

the name of the created plot file

`pyrad.graph.plots.plot_sun_retrieval_ts` (*sun_retrieval, data_type, fname*)
plots a time series

Parameters date : datetime object

time of the time series

value : float array

values of the time series

fname : str

name of the file where to store the plot

labelx : str

The label of the X axis

labely : str

The label of the Y axis

label1 : str

The label of the legend

titl : str

The figure title

period : float

measurement period in seconds used to compute accumulation. If 0 no accumulation is computed

Returns fname : str

the name of the created plot file

`pyrad.graph.plots.plot_timeseries` (*date, value, fname, labelx='Time [UTC]', labely='Value', label1='Sensor', titl='Time Series', period=0*)
plots a time series

Parameters date : datetime object

time of the time series

value : float array

values of the time series

fname : str

name of the file where to store the plot

labelx : str

The label of the X axis

labely : str

The label of the Y axis

label1 : str

The label of the legend

titl : str

The figure title

period : float

measurement period in seconds used to compute accumulation. If 0 no accumulation is computed

Returns fname : str

the name of the created plot file

`pyrad.graph.plots.plot_timeseries_comp` (*date1, value1, date2, value2, fname, labelx='Time [UTC]', labely='Value', label1='Sensor 1', label2='Sensor 2', titl='Time Series Comparison', period1=0, period2=0*)

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 : str

name of the file 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

Returns fname : str

the name of the created plot file

PYRAD.UTIL.RADAR_UTILS

Miscellaneous functions dealing with radar data

<code>create_sun_hits_field(rad_el, rad_az, ...)</code>	creates a sun hits field from the position and power of the sun hits
<code>create_sun_retrieval_field(par, imgcfg)</code>	creates a sun retrieval field from the retrieval parameters
<code>compute_quantiles_sweep(field, ray_start, ...)</code>	computes quantiles of a particular sweep
<code>compute_histogram_sweep(field, ray_start, ...)</code>	computes histogram of the data in a particular sweep

`pyrad.util.radar_utils.compute_histogram_sweep` (*field, ray_start, ray_end, field_name, step=None*)

computes histogram of the data in a particular sweep

Parameters **field** : ndarray 2D

the radar field

ray_start, ray_end : int

starting and ending ray indexes

field_name: str

name of the field

step : float

size of bin

Returns **bins** : float array

interval of each bin

values : float array

values at each bin

`pyrad.util.radar_utils.compute_quantiles_sweep` (*field, ray_start, ray_end, quantiles=None*)

computes quantiles of a particular sweep

Parameters **field** : ndarray 2D

the radar field

ray_start, ray_end : int

starting and ending ray indexes

quantiles: float array

list of quantiles to compute

Returns **quantiles** : float array

list of quantiles

values : float array

values at each quantile

`pyrad.util.radar_utils.create_sun_hits_field(rad_el, rad_az, sun_el, sun_az, data, imgcfg)`

creates a sun hits field from the position and power of the sun hits

Parameters **rad_el, rad_az, sun_el, sun_az** : ndarray 1D

azimuth and elevation of the radar and the sun respectively in degree

data : masked ndarray 1D

the sun hit data

imgcfg: dict

a dictionary specifying the ranges and resolution of the field to create

Returns **field** : masked ndarray 2D

the sun hit field

`pyrad.util.radar_utils.create_sun_retrieval_field(par, 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

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

p

`pyrad.flow.flow_control`, 1
`pyrad.graph.plots`, 42
`pyrad.io.io_aux`, 38
`pyrad.io.read_data_other`, 32
`pyrad.io.read_data_radar`, 28
`pyrad.io.write_data`, 35
`pyrad.proc.process_aux`, 5
`pyrad.proc.process_calib`, 19
`pyrad.proc.process_echoclass`, 8
`pyrad.proc.process_phase`, 10
`pyrad.proc.process_retrieve`, 15
`pyrad.prod.process_product`, 25
`pyrad.prod.product_aux`, 24
`pyrad.util.radar_utils`, 48

Symbols

[_add_dataset\(\)](#) (in module `pyrad.flow.flow_control`), 3
[_create_cfg_dict\(\)](#) (in module `pyrad.flow.flow_control`), 3
[_create_datacfg_dict\(\)](#) (in module `pyrad.flow.flow_control`), 3
[_create_dscfg_dict\(\)](#) (in module `pyrad.flow.flow_control`), 4
[_create_prdcfg_dict\(\)](#) (in module `pyrad.flow.flow_control`), 4
[_get_datasets_list\(\)](#) (in module `pyrad.flow.flow_control`), 4
[_get_datatype_list\(\)](#) (in module `pyrad.flow.flow_control`), 4
[_get_masterfile_list\(\)](#) (in module `pyrad.flow.flow_control`), 4
[_process_dataset\(\)](#) (in module `pyrad.flow.flow_control`), 5

C

[compute_histogram_sweep\(\)](#) (in module `pyrad.util.radar_utils`), 49
[compute_quantiles_sweep\(\)](#) (in module `pyrad.util.radar_utils`), 49
[create_sun_hits_field\(\)](#) (in module `pyrad.util.radar_utils`), 50
[create_sun_retrieval_field\(\)](#) (in module `pyrad.util.radar_utils`), 50

F

[find_cosmo_file\(\)](#) (in module `pyrad.io.io_aux`), 39
[find_rad4alpcosmo_file\(\)](#) (in module `pyrad.io.io_aux`), 39

G

[generate_field_name_str\(\)](#) (in module `pyrad.io.io_aux`), 40
[generate_sun_hits_products\(\)](#) (in module `pyrad.prod.process_product`), 27
[generate_timeseries_products\(\)](#) (in module `pyrad.prod.process_product`), 27
[generate_vol_products\(\)](#) (in module `pyrad.prod.process_product`), 27
[get_colobar_label\(\)](#) (in module `pyrad.graph.plots`), 43
[get_data\(\)](#) (in module `pyrad.io.read_data_radar`), 29

[get_data_rad4alp\(\)](#) (in module `pyrad.io.read_data_radar`), 29
[get_data_rainbow\(\)](#) (in module `pyrad.io.read_data_radar`), 30
[get_dataset_fields\(\)](#) (in module `pyrad.io.io_aux`), 40
[get_datatype_fields\(\)](#) (in module `pyrad.io.io_aux`), 40
[get_datatype_metranet\(\)](#) (in module `pyrad.io.io_aux`), 40
[get_datetime\(\)](#) (in module `pyrad.io.io_aux`), 41
[get_field_name\(\)](#) (in module `pyrad.graph.plots`), 43
[get_fieldname_pyart\(\)](#) (in module `pyrad.io.io_aux`), 41
[get_file_list\(\)](#) (in module `pyrad.io.io_aux`), 41
[get_process_type\(\)](#) (in module `pyrad.proc.process_aux`), 7
[get_product_type\(\)](#) (in module `pyrad.prod.product_aux`), 25
[get_save_dir\(\)](#) (in module `pyrad.io.io_aux`), 41
[get_sensor_data\(\)](#) (in module `pyrad.io.read_data_other`), 33

M

[main\(\)](#) (in module `pyrad.flow.flow_control`), 5
[make_filename\(\)](#) (in module `pyrad.io.io_aux`), 42
[merge_fields_cosmo\(\)](#) (in module `pyrad.io.read_data_radar`), 30
[merge_fields_rainbow\(\)](#) (in module `pyrad.io.read_data_radar`), 30
[merge_scans_cosmo\(\)](#) (in module `pyrad.io.read_data_radar`), 30
[merge_scans_cosmo_rad4alp\(\)](#) (in module `pyrad.io.read_data_radar`), 31
[merge_scans_rad4alp\(\)](#) (in module `pyrad.io.read_data_radar`), 31
[merge_scans_rainbow\(\)](#) (in module `pyrad.io.read_data_radar`), 31

P

[plot_bscopec\(\)](#) (in module `pyrad.graph.plots`), 43
[plot_cappi\(\)](#) (in module `pyrad.graph.plots`), 44
[plot_histogram\(\)](#) (in module `pyrad.graph.plots`), 44
[plot_ppi\(\)](#) (in module `pyrad.graph.plots`), 45
[plot_quantiles\(\)](#) (in module `pyrad.graph.plots`), 45
[plot_rhi\(\)](#) (in module `pyrad.graph.plots`), 46

plot_sun_hits() (in module pyrad.graph.plots), 46
plot_sun_retrieval_ts() (in module pyrad.graph.plots), 47
plot_timeseries() (in module pyrad.graph.plots), 47
plot_timeseries_comp() (in module pyrad.graph.plots), 48
process_attenuation() (in module pyrad.proc.process_phase), 11
process_cdr() (in module pyrad.proc.process_retrieve), 17
process_correct_bias() (in module pyrad.proc.process_calib), 21
process_correct_noise_rho_hv() (in module pyrad.proc.process_calib), 21
process_correct_phidp0() (in module pyrad.proc.process_phase), 11
process_echo_filter() (in module pyrad.proc.process_echoclass), 9
process_echo_id() (in module pyrad.proc.process_echoclass), 9
process_estimate_phidp0() (in module pyrad.proc.process_phase), 12
process_filter_snr() (in module pyrad.proc.process_echoclass), 9
process_hydroclass() (in module pyrad.proc.process_echoclass), 10
process_kdp_least_square_double_window() (in module pyrad.proc.process_phase), 12
process_kdp_least_square_single_window() (in module pyrad.proc.process_phase), 13
process_l0() (in module pyrad.proc.process_retrieve), 17
process_phidp_kdp_lp() (in module pyrad.proc.process_phase), 14
process_phidp_kdp_Maesaka() (in module pyrad.proc.process_phase), 13
process_point_measurement() (in module pyrad.proc.process_aux), 7
process_rainrate() (in module pyrad.proc.process_retrieve), 17
process_raw() (in module pyrad.proc.process_aux), 8
process_rho_hv_rain() (in module pyrad.proc.process_calib), 22
process_save_radar() (in module pyrad.proc.process_aux), 8
process_selfconsistency_bias() (in module pyrad.proc.process_calib), 22
process_selfconsistency_kdp_phidp() (in module pyrad.proc.process_calib), 22
process_signal_power() (in module pyrad.proc.process_retrieve), 18
process_smooth_phidp_double_window() (in module pyrad.proc.process_phase), 14
process_smooth_phidp_single_window() (in module pyrad.proc.process_phase), 15
process_snr() (in module pyrad.proc.process_retrieve), 18
process_sun_hits() (in module pyrad.proc.process_calib), 23

pyrad.flow.flow_control (module), 1
pyrad.graph.plots (module), 42
pyrad.io.io_aux (module), 38
pyrad.io.read_data_other (module), 32
pyrad.io.read_data_radar (module), 28
pyrad.io.write_data (module), 35
pyrad.proc.process_aux (module), 5
pyrad.proc.process_calib (module), 19
pyrad.proc.process_echoclass (module), 8
pyrad.proc.process_phase (module), 10
pyrad.proc.process_retrieve (module), 15
pyrad.prod.process_product (module), 25
pyrad.prod.product_aux (module), 24
pyrad.util.radar_utils (module), 48

R

read_disdro_scattering() (in module pyrad.io.read_data_other), 33
read_rad4alp_cosmo() (in module pyrad.io.read_data_other), 33
read_selfconsistency() (in module pyrad.io.read_data_other), 34
read_smn() (in module pyrad.io.read_data_other), 34
read_status() (in module pyrad.io.read_data_other), 34
read_sun_hits() (in module pyrad.io.read_data_other), 34
read_sun_hits_multiple_days() (in module pyrad.io.read_data_other), 34
read_sun_retrieval() (in module pyrad.io.read_data_other), 35
read_timeseries() (in module pyrad.io.read_data_other), 35

W

write_sun_hits() (in module pyrad.io.write_data), 37
write_sun_retrieval() (in module pyrad.io.write_data), 37
write_timeseries() (in module pyrad.io.write_data), 37