RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

METEO FRANCE

Pyrad architecture and principles

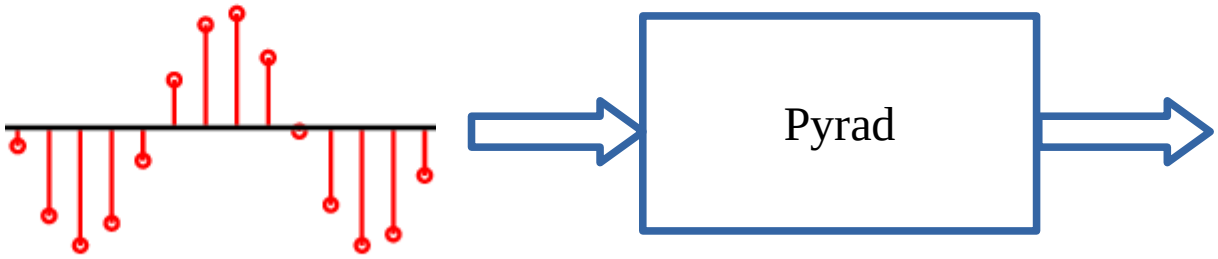
Jordi Figueras i Ventura
Pyrad course

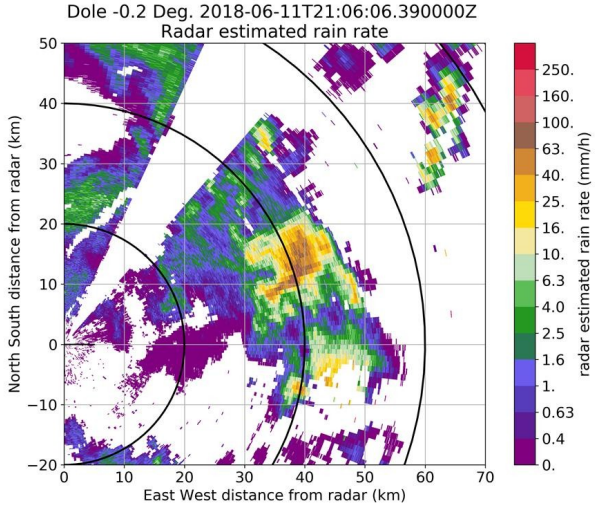
Contents

1. Introduction

What is Pyrad ?

Pyrad is a processing framework that allows the creation of flexible and replicable data processing chains with no programming. It is capable of operating in **real time** or **off-line.** It is aimed mainly for weather radar data processing but has some limited functionality allowing to process data from other sensors.





History and use

2016 – Initially developed at MeteoSwiss as Python replacement for and IDL processing framework

2020 – First release as a python and conda package (v0.4.4)

2021 – Agreement to have a joined development between MeteoSwiss and Météo-France

2022 – Version 1.0.0

Use :

- Operational real-time post-processing of X-band radar data (MeteoSwiss)
- Operational polarimetric data quality monitoring (MeteoSwiss)
- Platform for rapid algorithm prototyping
- Platform for MeteoSwiss and Météo-France radar volume data processing and visualization by third parties
- Common development platform for institutions partnering with MeteoSwiss or Météo-France

General characteristics

- Python-based (> v. 3.7)
- Linux platform
- Open source, version controlled (**https://github.com/MeteoSwiss/pyrad**)
- Core based on our own version of **ARM-DOE Py-ART** (The Pyrad project contributes back regularly)
- Possibility to ingest data from multiple radars
- Ingests multiple data types: IQ data, spectral data, polarimetric and Doppler moments, Cartesian data, etc.
- Capable of reading the main file formats used for volume radar data storage
- **Automatic documentation published online** based on doc-strings
- Easy to install (PyPI, conda)

2. Pyrad working philosophy

General characteristics

KEY CONCEPT : Separation between **dataset** generation and **product** generation

Dataset : New data generated from the processing of a Pyrad data object. It can be in the form of a new field of the same object type or a completely new object. It can be re-ingested in the data processing chain. Examples :

- rainfall rate field generated from a reflectivity field contained in a radar object

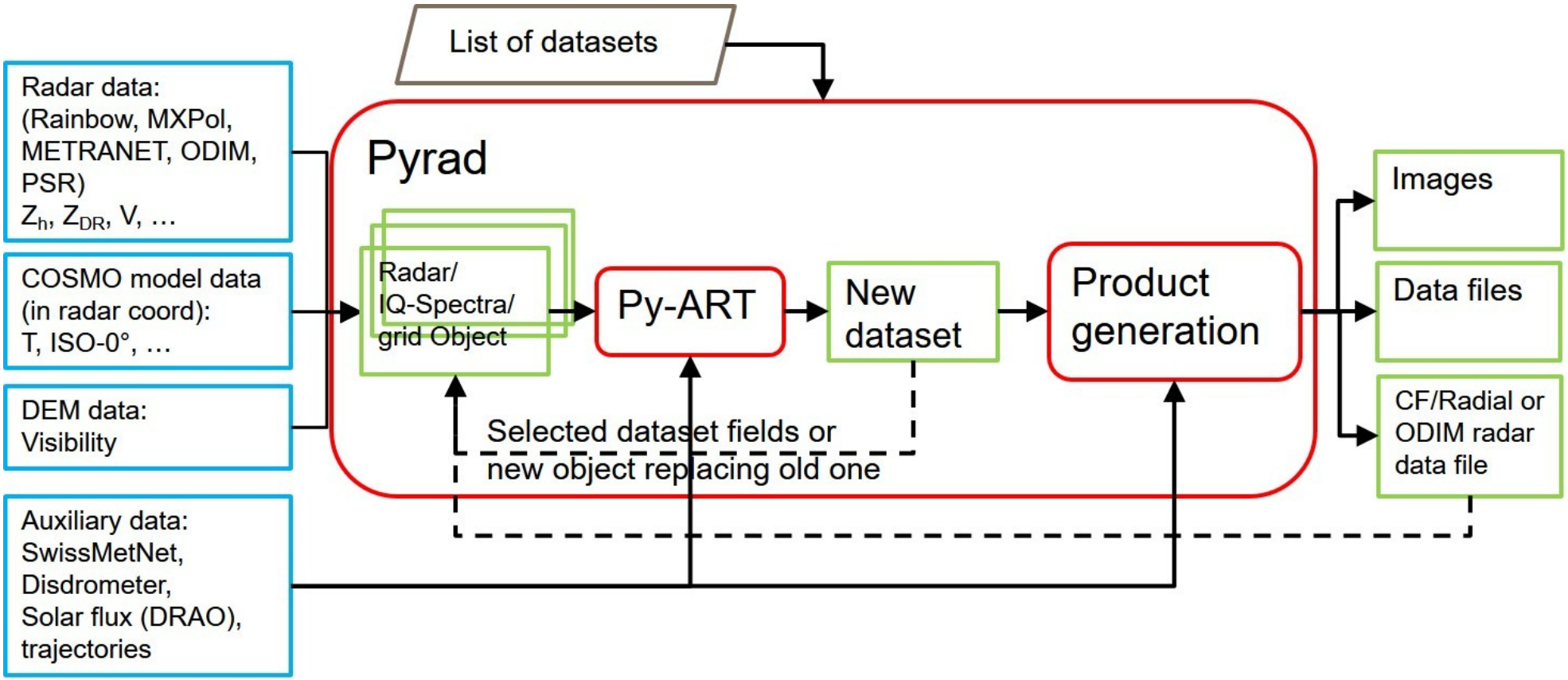
- Reflectivity generated from a spectral data object

Product : Output generated out of a dataset for human or machine consumption. Examples :

- PPI of reflectivity

- File containing timeseries of values at a point of interest

Pyrad flow diagram



Configuration files

Main config file	• Base paths of data • Location of config files • Base path for output data and image format(s)
location config file	• Radar(s) name and scan list • General radar and scan characteristics (scan periodicity, radar constant, …) • Images configuration

product config file	• List of datasets to generate
	• For each dataset
	  • List of inputs
	  • Dataset specific configuration
	  • List of products to generate
	  • MAKE_GLOBAL
	  • SUBSTITUTE_OBJECT
	  • FIELDS_TO_REMOVE

Internally all the configuration parameters are stored in a dictionary cfg
From cfg pyrad creates :
- datacfg : necessary parameters to read the input data
- dscfg : parameters to create the datasets (one per dataset)
- prdcfg : parameters to create products (one per product)

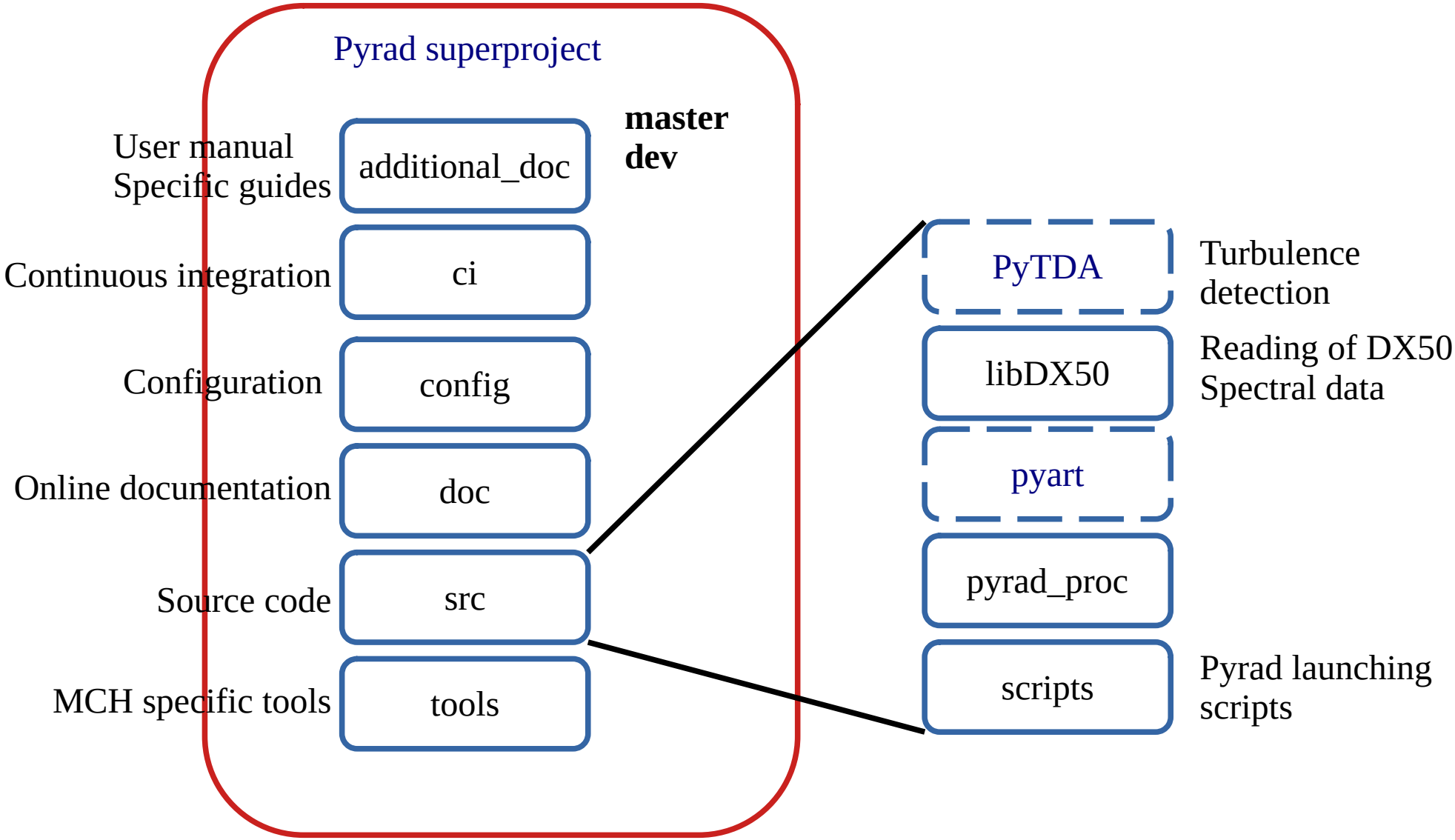
Config file data types

Data types in the config file are mapped to Python values

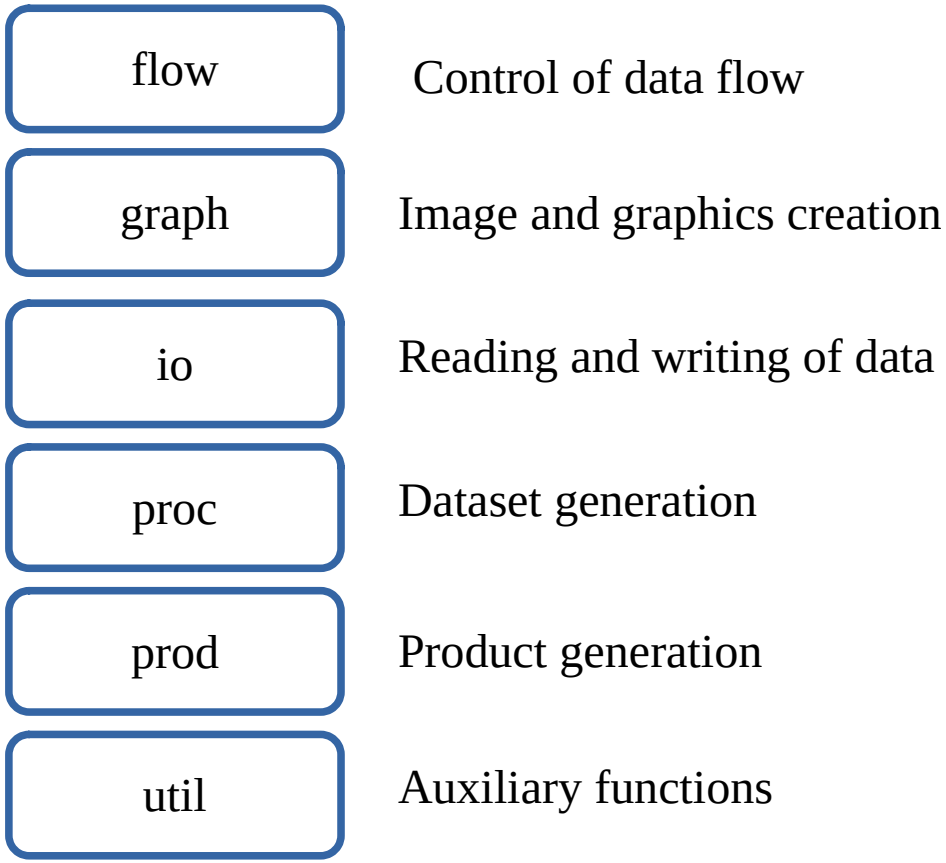
Pyrad data type Keyword	Python data type
BYTE, INT, LONG, HEX	int
EXP, FLOAT, DOUBLE	float
STRING	str
BYTARR, INTARR, LONARR, HEXARR	Numpy array of byte, int32, int64, H
EXPARR, FLTARR, DBLARR	Numpy array of float64, float64, double
STRARR	List of strings
STRUCT	dict

3. Pyrad architecture

Github architecture



pyrad_proc modules



Proc files

process_aux.py	Selection of process according to config file. Radar data resampling. Conversion from radar to grid data
process_calib.py	Bias correction. Monitoring with the sun and ground clutter
process_cosmo.py	Interpolation of NWP data into radar grid
process_dem.py	Managing of Digital Elevation Models (e.g. visibility)
process_Doppler.py	De-aliasing, turbulence detection, wind retrievals
process_echoclass.py	Echo classification and data filtering
process_grid.py	Grid data processing
process_intercomp.py	Radar-radar intercomparison
process_iq.py	IQ data processing. Computation of moments from IQ data
process_monitoring.py	Polarimetric data quality monitoring
process_phase.py	PhiDP and KDP retrievals. Attenuation correction
process_retrieve.py	Miscellaneous retrievals. (e.g. VPR, rain rate, noise, etc.)
process_spectra.py	Spectral processing. Computation of moments from spectral data
process_timeseries.py	Time series (POI, QVP, columns at a prescribed location, etc.)
process_traj.py	Trajectory processing (retrieval of data at given locations)

Prod files

process_grid_products.py	Products from grid datasets
process_intercomp_products.py	Products from intercomp datasets
process_monitoring_products.py	Products from monitoring datasets
process_product.py	Products from miscellaneous datasets (e.g. QVP, ML, VPR, NWP models, etc.)
process_spectra_products.py	Products from spectra datasets
process_timeseries_products.py	Products from timeseries (includes trajectory on a CAPPI)
process_traj_products.py	Products from trajectory
process_vol_products.py	Products from radar volume datasets
product_aux.py	Selection of product family according to config file

4. Launching Pyrad

Pyrad processing status

Status 0 : Initialization of datasets

Status 1 : Sequential processing of input data. Persistent data and parameters are stored internally

Status 2 : All input data consumed. Final dataset production if necessary

Pyrad Launching scripts

main_process_data.py	Process data sequentially according to starttime and endtime defined in command line or by input file
main_process_data_period.py	Process available data within time intervals specified by user between dates specified by user. Useful to obtain daily statistics
main_process_data_rt.py	Real time data processing

Trajectory processing with main_process_data.py

If a trajfile is provided pyrad enters in a special processing mode

-t, --trajfile	Trajectory file or file containing disjoint processing periods
--trajtype	plane : File contains a trajectory defined by lat, lon, altitude and time. Pyrad will look for data inputs within the time range defined by the file and a time series of radar data at the location of the « plane » can be obtained lightning : As above but with multiple simultaneous trajectories proc_periods : File contains a series of disjoint processing periods. Useful to process only data at periods of interest
--flashnr	If trajtype is lightning you can select the trajectory that you want to process. If 0 all trajectories will be processed

5. File management

Reading input data files

- **Off-line:** Pyrad looks for all files present in the expected file directories within a time period defined by the user or the --trajfile
- **Real time:** Pyrad looks for new files that have appeared between the last time it processed a file and the current time, then it processes all new files sequentially
- If more than one type of file must be read per time step, a **master file** is defined and all other files depend on this master
- Metadata required for data processing is read from the file. If metadata is absent or wrong the user can specify it in the loc config file
- Internally, all data and metadata is stored in a single data object. Therefore the data read in the files must be compatible with the same object type
- If the input data does not have the same grid, Pyrad will internally **interpolate the data to the grid** of the master file(s)
- If the user is interested only in a particular region it can specify the limits of the region in the loc file

Reading input data files. Searching for data

Pyrad looks for data in the following base repositories :

datapath : primary base path where to look for data

loadbasepath : base path where to look for datasets generated by pyrad

cosmopath : base path where to look for model data

dempath : base path where to look for DEM data (e.g. visibility)

satpath : base path where to look for satellite data in gridded netcdf format

psrpath : base path where to look for spectral data (needs datapath)

iqpath : base path where to look for IQ data (needs datapath)

Reading input data files. The ScanList keyword

A radar volume may be distributed in separate files by **sweep (fixed angle), datatype or both**

The **ScanList** variable consists on a list of identifiers of each file in order to be able to group them in a single radar volume

The first identifier in the list will be used as **master** and all the others will be referred to it

ScanList STRARR 9
- *dBuZ_PAMA40LFPW*
- *dBuZ_PAMB40LFPW*
- *dBuZ_PAMC40LFPW*
- *RhoHV_PAMA40LFPW*
- *RhoHV_PAMB40LFPW*
- *RhoHV_PAMC40LFPW*
- *ZDR_PAMA40LFPW*
- *ZDR_PAMB40LFPW*
- *ZDR_PAMC40LFPW*

ScanList STRARR 9
- *MALS_LOC_274_up_nopsr.ele/*
- *MALS_LOC_278_dw_nopsr.ele/*
- *MALS_LOC_282_up_nopsr.ele/*
- *MALS_LOC_286_dw_nopsr.ele/*
- *MALS_LOC_284_up_nopsr.ele/*
- *MALS_LOC_280_dw_nopsr.ele/*
- *MALS_LOC_276_up_nopsr.ele/*
- *MALS_LOC_272_dw_nopsr.ele/*
- *MALS_LOC_268_up_nopsr.ele/*

ScanList STRARR 5
- *001*
- *002*
- *003*
- *004*
- *005*

Folder hierarchy : path_convention keyword

Pyrad can work with multiple folder hierarchies depending on the input data and the user-specified data format. The data of the files has to be compatible (e.g. do not mix grid data with radar volume data).

Most common file formats folder hierarchy:

ODIM, ODIMBIRDS, CFRADIAL, CFRADIAL2, CF1, NEXRADII, GAMIC, ODIMGRID :

- MCH : *datapath/[YYJJJ]/[(M/P)(L/H)(X)YYJJJ]/*
- ODIM :*datapath/[day_dir]/*
- RT : *datapath/[(M/P)(L/H)(X)]/*

Pyrad output (**CFRADIALPYRAD, ODIMPYRAD, PYRADGRID, ODIMPYRADGRID, NETCDFSPECTRA, CSV**):

loadbasepath/loadname/[YYYY-MM-DD]/dataset/product/

Folder hierarchy : path_convention keyword

Other formats convention hierarchy:

RAINBOW : *datapath/[scan][YYYY-MM-DD]*

RAD4ALP :

- LTE : *datapath/[(M/P)(L/H)(X)YYhdfJJJ]/*
- MCH : *datapath/[YYJJJ]/[(M/P)(L/H)(X)YYJJJ]/*
- RT : *datapath/[(M/P)(L/H)(X)]/*

RAD4ALPGRID, RAD4ALPGIF, RAD4ALPBIN

SATGRID : *satpath/[YYYY]/[MM]/[DD]/*

MFCFRADIAL, MFBIN, MFPNG, MFGRIB, MFDAT, MFCF : *datapath/[day_dir]/*

MXPol : Default and LTE

COSMORAW (needs datatype as reference):

- *cosmopath/raw[1]/*
- MCH : *cosmopath/raw[1]/[YYYY-MM-DD]/*

Reading data files: the *datatype* keyword

The ***datatype*** string allows to define which field must be read and from which file type and into which structure has to be stored. Internally all data types in the product config file are grouped so that reading is done only once.

Datatype definition has the following structure:

datatype STRING [radarnr]:[datagroup],[fieldname],[dataset/file_naming],[product]

Example:

datatype STRARR 2

RADAR001:ODIM,dBZ,D{%Y/%m/%d}-F{%Y%m%d%H%M%S}

RADAR001:ODIMPyrad,hydro,hydrometeor_classification,SAVED_VOLUME

Reading data files : the datatype keyword

radarnr	RADARXXX : The number of the data object where the input data will be stored. Starting by 001. If more than one data object will be input this keyword is used to identify to which data object the metadata belongs to (e.g. in the ScanList)	RADAR001 is the default. There is no need to specify it if only one radar will be used
datagroup	Type of input file	RAINBOW is the default.
fieldname	Fieldname according to Pyrad convention	e.g. *dBZ*
dataset/ file_naming	Dataset=> For pyrad output files: Name of the dataset subdirectory where data has been stored. file_naming=> For files following the ODIM path convention: The directory structure linked to the date and the file time stamp format	Dataset=> e.g. *Zh* file_naming=> *D{format for directory}-F{format for file}* *e.g. D{%Y/%m/%d}-F{%Y%m%d%H%M}*
product	For pyrad output files: Name of the product subdirectory where data has been stored	e.g. *PPI_EL001*

Datagroups for radar volume object. Widely used formats

ODIM	HDF5 European standard
ODIMPYRAD	As above but with the Pyrad folder hierarchy
CFRADIAL	CF Radial V1 format
CFRADIALPYRAD	As above but with the Pyrad folder hierarchy
CFRADIAL2	CF Radial V2 format. Only partially compatible
NEXRADII	Nexrad file format
PROC	Data generated on the fly by Pyrad

Datagroups for radar volume object. Proprietary formats

ODIMBIRDS	HDF5 for profile data used by the aeroecology community. Similar to ODIM but with additional metadata
CFRADIALCOSMO	NWP data in radar coordinates in a CF Radial V1 format
MFCFRADIAL	CF Radial V1 format with MF fieldnames mapping
RAINBOW	Leonardo proprietary Rainbow file format
COSMO	NWP data in radar coordinates in Rainbow format
DEM	Visibility data in Rainbow format
GAMIC	GAMIC proprietary file format
MXPOL	MXPOL proprietary file format
RAD4ALP	METRANET format (ELDES proprietary format used at MeteoSwiss)
RAD4ALPCOSMO	NWP data in radar coordinates in METRANET format
RAD4ALPHYDRO	Hydrometeor classification data in METRANET format. Needs a RAD4ALP file to get the metadata
RAD4ALPDOPPLER	De-aliased Doppler velocity in METRANET format. Needs a RAD4ALP file to get the metadata
CF1	Data in the NETCDF/CF format. Data format of a cloud radar

All formats read by Py-ART can be easily added to Pyrad

Datagroups for grid object

ODIMGRID	Cartesian grid products in ODIM file format
ODIMPYRADGRID	Pyrad generated Cartesian data in ODIM format
PYRADGRID	Pyrad generated Cartesian data in NETCDF format
RAD4ALPGRID	METRANET format for Cartesian grid products
RAD4ALPGIF	MeteoSwiss data in GIF format
RAD4ALPBIN	MeteoSwiss data in binary format
MFBIN	Météo-France data in binary format
MFDAT	Météo-France data in text format
MFPNG	Météo-France data in PNG format
MFGRIB	Météo-France data in GRIB format
MFCF	Météo-France data using NETCDF/CF format
SATGRID	Satellite data in NETCDF format

Datagroups for IQ/Spectra object

RAD4ALPIQ	Internal MeteoSwiss data format for IQ data (binary file). Requires reading a radar object to get the metadata
PSRSPECTRA	Rainbow file format for spectral data. Requires reading a radar object to get the metadata
NETCDFSPECTRA	Pyrad output of the IQ/Spectra object. Written in NETCDF in a format analogous to CFRadial1

Other pathes where pyrad searches for data and metadata

configpath: basepath for configuration files required for specific dataset or product generation routines

General pathes to Auxiliary data used for specific dataset or product generation functions:

colocgatespath: basepath to files containing the position of co-located gates used to inter-compare radar output

solarfluxpath: basepath to location of a solar flux file produced by DRAO

excessgatespath: basepath to location of file containing the position of gates used for clutter monitoring

disdropath: basepath to disdrometer data. Used to compare time series of radar data and disdrometer data at the disdrometer location

smnpath: basepath to weather station data. Used to compare time series of radar-derived vs rain gauge measured rainfall rate at the location of the rain gauge

Other dataset/product generation functions may require auxiliary data defined at the dataset

Pyrad output folder structure

By default Pyrad will store the data using the following folder structure:

saveimgbasepath/[proc_space]/[time_info]/[dataset_name]/[product_name]/

e.g. home/jfigui/TOUL/2023-04-25/dBZ/PPI_EL001/

saveimgbasepath	Base path where Pyrad output data is stored	e.g. */home/jfigui/*
proc_space	Where to store all data output generated from the same main config file. Defined by the *name* keyword in the main config file	e.g. *TOUL*
time_info	Usually YYYY-MM-DD. Can be absent depending on the product. e.g. Long term time series	Dataset=> e.g. *Zh* file_naming=> *D{directory format}-F{fileformat}* *D{%Y%m%d}-F{%Y%m%d%H%M}*
dataset_name	Name of the dataset as defined by the product config file	Can be modified using keyword ***dssavename*** when defining the dataset
product_name	Name of the output product as defined by the product config file	Can be modified using keyword ***prdsavedir*** when defining the product

Pyrad output file name convention

Usually the Pyrad output file names have the following structure:

[time_stamp]_[run_info]_[product_id]_[dataset_id]_[fieldname]_[prod_info].[termination]

time_stamp	time stamp info. Usually the nominal time stamp as read from the input file. Usually of the form YYYYmmddHHMMSS	e.g. 20230425120500
run_info	Additional information provided at run time by the keyword *-i, --infostr* in the launching script	e.g. *run23*
product_id	String identifying the product	*e.g. ppi*
dataset_id	String identifying the dataset type	*e.g. RAW* (output of raw data reading)
fieldname	Pyrad convention field name	e.g. *dBZ*
prod_info	Additional string identifying the product.	e.g. *el0.4*
termination	For images it can be that of any file format accepted by Matplotlib and it is defined by keyword ***imgformat*** in the main config file. Outputs that are not grid or radar objects usually are *.csv*	

6. Constructing the processing chain

Constructing the processing chain

- Each dataset type is identified by a **keyword** that internally is associated to a **processing function**
- Internally the datasets are grouped in **families**. Those families can generate similar products
- The user can define up to 99 **levels of processing** for each **input data object**
- Datasets that are not inter-dependent can be generated at the same processing level. There is an option to generate those in parallel
- Levels will be processed sequentially. At the end of each processing level the keywords ***MAKE_GLOBAL, SUBSTITUTE_OBJECT*** and ***FIELDS_TO_REMOVE*** will control the behaviour of the new dataset created
- For each new dataset as many **products** as specified by the user will be generated. There is an option to generate those in parallel. If desired no product needs to be generated

Constructing the processing chain

MAKE_GLOBAL	1 - The newly generated dataset fields will be added to the data object and will be available for the next processing level. Assumes that the generated dataset is compatible with the current data object. If a field with the same name exists it will be overwritten 0 – The dataset will not be added to the data object
FIELDS_TO_REMOVE	List of fields to remove. The fields removed will not be available for the next processing level. Useful to remove intermediate fields to reduce the memory footprint
SUBSTITUTE_OBJECT	1 - The data object used at the current level will be erased and substituted by the newly generated dataset. Useful for e.g. transition from a volume radar object to a grid

Dataset families. Most common

VOL	generate_vol_products	Radar volume output
TIMEAVG	generate_time_avg_products	Radar volume time averaging products. Same products as above but with different time stamp
SPECTRA	generate_spectra_products	Spectra or IQ volume output
GRID	generate_grid_products	Cartesian grid data ouput
GRID_TIMEAVG	generate_grid_time_avg_products	Time-averaging or accumulation of grid data. Same products as above but with different time stamp
QVP	generate_qvp_products	Time-column data (e.g. QVP, columns, etc.)
TIMESERIES	generate_timeseries_products	Time series of POIs products

Dataset families. Specialized functions

MONITORING	generate_monitoring_products	Polar data monitoring
SUN_HITS	generate_sun_hits_products	Monitoring using the sun
OCCURRENCE	generate_occurrence_products	Monitoring using ground clutter
COLOCATED_GATES	generate_colocated_gates_products	Data used to find co-located gates for radar inter-comparison
INTERCOMP	generate_intercomp_products	Inter-comparison between radars
COSMO2RADAR	generate_cosmo_to_radar_products	Output of the interpolation of NWP data into radar coordinates
COSMO_COORD	generate_cosmo_coord_products	Data used to select the grid coordinates of the NWP data that will be interpolated into radar coordinates
ML	generate_ml_products	Melting layer detection data
VPR	generate_vpr_products	Vertical profile of reflectivity correction data
CENTROIDS	generate_centroids_products	Data used to obtain centroids for the semi-supervised hydrometeor classification
TRAJ_ONLY	generate_traj_product	Trajectory products

Identification of data fields

- Internally pyrad uses the naming convention of Py-ART which follows closely the **CF/Radial convention** for radar data fields.
- There are some fields that are non-standard and therefore defined by Pyrad
- The Py-ART names used internally are defined in the **Py-ART config file**
- Since the Py-ART naming is very long, Pyrad uses short keywords in the Pyrad config files. The short keywords are mapped internally using the function get_fieldname_pyart in io_aux.py
- The special keyword « all_fields » allows to load all data saved in a Pyrad generated volume

Generation of images

- Standard radar images (PPI, CAPPI, RHI, etc.) are generated using the **plotting functions from Py-ART**
- Standard Cartesian images (Surface, cross-sections, etc.) are generated using the **plotting functions from Py-ART**
- Other images (e.g. B-scope) and graphics (e.g. time series) are generated by Pyrad itself
- Colormap and value limits, field name to show, units, etc. are generally defined at the **Py-ART config file**. Pyrad allows using a discrete colormap
- The image size, resolution and area plotted are defined by ***ImageConfig*** structs in the pyrad loc config file
- Images can be overplot on a map generated by cartopy

Generation of images

ppiImageConfig	PPI-like image configuration
rhiImageConfig	RHI-like image configuration
ppiMapImageConfig	PPI-like image overplotted on a map configuration
gridMapImageConfig	Cartesian grid image overplot on a map configuration
sunhitsImageConfig	Sun hits image (delta_az, delta_ele)

7. Config files: an example


Thank you!
Grazie mille!
Moltes Gràcies!
Merci!