|  |
| --- |
|  |

**PYRAD**

User Manual

|  |
| --- |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | In progress | In validation | Approved |
| Document status |  |  |  |

Version 0.1

Document History

Responsible People

|  |  |
| --- | --- |
| Creation/Edition | Jordi Figueras i Ventura (fvj) |
| Revision |  |
| Approval |  |
| Further information |  |

Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Edited by | Date | Activity |
| 0.1 | fvj | 28.06.2016 | Creation |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[Chapter 1 Installation 4](#_Toc457293380)

[1.1 Py-ART installation instructions 4](#_Toc457293381)

[1.1.1 Pre-installation requisites 4](#_Toc457293382)

[1.1.2 Py-ART installation 5](#_Toc457293383)

[1.1.3 Py-ART extensions 6](#_Toc457293384)

[1.2 Pyrad\_proc installation instructions 7](#_Toc457293385)

[Chapter 2 Developing pyrad 8](#_Toc457293386)

[2.1 The pyrad git architecture 8](#_Toc457293387)

[2.2 Code style 8](#_Toc457293388)

[2.3 Developing the pyrad git superproject 8](#_Toc457293389)

[2.4 Dealing with git submodules 10](#_Toc457293390)

[2.4.1 Installing a git submodule 10](#_Toc457293391)

[2.4.2 Updating the local submodule working branch with changes in the master public library 10](#_Toc457293392)

[2.4.3 Transferring changes from the local submodule working branch to the master public library 11](#_Toc457293393)

[Chapter 3 References 13](#_Toc457293394)

Figures

[Fig. 1 Git flow diagram 9](#_Toc457293395)

Tables

**No table of figures entries found.**

# Installation

## Py-ART installation instructions

### Pre-installation requisites

**Note 1:** If you are using the zueub222 MeteoSwiss server the package pre-installation has already been performed for you. Simply add the required commands in your .bashrc file (in red in this section).

**Note 2:** The pathes in the .bashrc file here are those for zueub222. If you are working in another server modify them accordingly

A version of Anaconda supporting Python 3.5 or higher should be installed in the server. It can be found in [1]. Do not forget to add the path to Anaconda in your .bashrc file. In the case of zueub222 is:

export PATH=/opt/anaconda3/bin/:$PATH

The following default packages in the Anaconda installation are necessary to run Py-ART: NumPy, SciPy and matplotlib. Before installing additional packages, depending on the configuration of your server, you may need to switch off ssl verification:

conda config --set ssl\_verify false

On addition, the package netCDF4 has to be installed using the following command:

conda install netcdf4

There are a number of optional dependencies available from conda that can be installed as well: h5py (to read HDF5 files), basemap (to plot grids on geographic maps), nose (to run Py-ART unit tests), gdal (to output GeoTIFFS from grid objects) and some sort of linear programming solver to use the LP phase processing method such as CyLP. They can all be installed simply with the command:

conda install basemap nose h5py gdal

In addition the TRMM Radar Software Library (RSL) (See for the webpage [2]) can be installed to read radar files in particular formats. The installation is performed from the jjhelmus channel:

conda install -c <https://conda.binstar.org/jjhelmus> trmm\_rsl

**WARNING:** Be aware that there are other versions of the library in other channels but, if installed with conda, only this channel should be used because otherwise the library is not working properly due to issues with the pathes.

The location of the library (where the lib and include directories are) should be specified with the following command (typically on your .bashrc file):

export RSL\_PATH=/opt/anaconda3/

Finally, wradlib (see [3]) is used to compute the texture of a differential phase field. It can be installed from conda adding the conda-forge channel:

conda config --add channels conda-forge

And then installing the package:

conda install wradlib

Finally, to have the capacity to read rainbow5 files you have to add the xmltodict library:

pip install --trusted-host pypi.python.org xmltodict

The location of the GDAL data has to be specified by writing in your .bashrc file the following command:

export GDAL\_DATA=/opt/anaconda3/share/gdal

**WARNING:** Additional packages or different versions of the packages may be necessary to run wradlib property. You can check if the installation has been performed properly by typing

python

and then:

import wradlib

If there is any error you should download the missing library. A good practice may be to update all the installed packages in anaconda by typing:

conda update --all

In addition to the standard Py-ART packages at MeteoSwiss we have created specific libraries to read rad4alp and DX50 radar data. For the rad4alp data, make sure that you have access to the library srn\_idl\_py\_lib.[machine].so and add the path to your .bashrc:

export METRANETLIB\_PATH=/proj/lom/idl/lib/radlib4/

For the rainbow data files (i.e. the file type used by the DX50) there exists a reader in the wradlib library but we have also created an alternative reader using a c-based library which is located in the libRainbow directory of the pyrad repository. To use that one add the path to the Python pathes in your .bashrc. For example:

export PYTHONPATH=[Pyrad\_path]/src/libRainbow/:$PYTHONPATH

### Py-ART installation

**Note:** Make sure to have the latest version of the pyrad repository in your local server.

Py-ART repository can be found on [4]. A compiled version is available from the conda repository:

conda install -c <https://conda.anaconda.org/jjhelmus> pyart

We have our own (modified) version of py-ART in the pyrad repository. To install Py-ART in your personal repository enter into the directory pyart-master and simply type:

python setup.py install --user

Optionally, if you have the rights for this you can install it for all users by typing:

python setup.py build

sudo python setup.py install

To check whether the library dependences have been installed properly type:

python -c "import pyart; pyart.\_debug\_info()"

Py-ART has a default config file called default\_config.py and located in folder pyart. If you would like to work with a different config file you have to specify the location in the variable PYART\_CONFIG in your .bashrc file. For example:

export PYART\_CONFIG= [Pyrad\_path]/config/pyart/mch\_config.py

The Pyrad library has its own config file in the aforementioned path.

### Py-ART extensions

Several extensions build over Py-ART are available. In the following we will show how to install the ones available in the pyrad repository.

#### ARTView

ARTView is an interactive radar viewing browser. The source code can be found in [5]. The simplest way to install it is using conda:

conda install -c jjhelmus artview

#### DualPol

DualPol is a package that facilitates dual-polarization data processing. Its source code can be found in [6]. Apart from Py-ART it is built on the libraries CSU\_RadarTools and SkewT, which have to be installed first.

SkewT can be found in [7]. It provides a set of tool for plotting and analysis of atmospheric data. To install it simply download the source code, go to the main directory and type:

python setup.py install

CSU\_RadarTools can be found in [8]. It provides a set of tools to process polarimetric radar data developed by the Colorado State University. To install it simply download the source code, go to the main directory and type:

python setup.py install

Finally you can install DualPol by downloading the source code, going to the main directory and typing:

python setup.py install

#### PyTDA

PyTDA is a package that provides functions to estimate turbulence from Doppler radar data. Its source code can be found in [9]. To install it simply download the source code, go to the main directory and type:

python setup.py install

#### SingleDop

SingleDop is a package that retrieves two-dimensional low-level winds from Doppler radar data. It can be found in [10]. It requires PyTDA to be installed in order to work. To install it simply download the source code, go to the main directory and type:

python setup.py install

#### PyBlock

PyBlock estimates partial beam blockage using methodologies base on the self-consistency of polarimetric radar variables in rain. It can be found in [11]. It requires DualPol to be installed in order to work properly. To install it simply download the source code, go to the main directory and type:

python setup.py install

## Pyrad\_proc installation instructions

Pyrad\_proc is the container for the MeteoSwiss radar processing framework. The core radar processing functions are based on Py-ART. Therefore Py-ART should be correctly installed before running Pyrad\_proc.

To install pyrad\_proc, simply go to the main directory and type:

python setup.py install --user

The main program is in the folder pyrad\_proc/scripts/ and it is called main\_process\_data.py. To run the program simply enter:

python main\_process\_data.py [config\_file] [process\_start\_time] [process\_end\_time]

For example:

python main\_process\_data.py '/home/lom/users/fvj/pyrad/config/processing/paradiso\_fvj\_vol.txt' '20140523000000' '20140523001000'

# Developing pyrad

## The pyrad git architecture

## Code style

Pyrad and its submodels follow the PEP8 standard [13]. To make sure that your code formally complies with the standard make use of the pycodestyle tool [14]. You can install it from conda simply typing:

conda install –c conda-forge pycodestyle=2.0.0

For users of zueub222 the package has already been installed. Detailed instructions on how to use the tool can be found in [15]. The simplest use is simply to type:

pycodestyle [your\_file.py]

A list of errors and their location will appear.

## Developing the pyrad git superproject

The pyrad git superproject repository is stored in the internal MeteoSwiss gitlab in [12][.](https://gitlab.meteoswiss.ch/MDR/pyrad) As a developer the typical git commands apply so if you want to collaborate and you have the right to access the repository simply type:

git clone [gitlab@gitlab.meteoswiss.ch:MDR/pyrad.git](mailto:gitlab@gitlab.meteoswiss.ch:MDR/pyrad.git)

To get a copy of the project in your working file. The regular commands summarized in Fig. 1 apply. However one has to remember that the changes in the submodules have to be pushed first to the submodule repository before commiting the super-project.

To push changes in the submodule go to the main folder of the submodule and do the following:

1. Check the status of the module:

git status

1. Check to which remote you are connected:

git remote –v

1. Check in which branch are you working in (for regular Py-ART developers should be pyart-mch)

git branch

1. If the branch is not the desired one change it:

git checkout pyart-mch

1. Add or remove the files you want to commit with the regular commands git add git rm.
2. Commit your changes:

git commit –a –m “explanation of my changes”

1. Pull the remote and deal with possible conflicts. If necessary commit again:

git pull

1. Once satisfied, push the changes to the remote:

git push

You will be asked to input your user name and password.

Once this is done, you can push the changes in the super-project by going to the main folder of the super-project and repeating steps 4 to 7. Do not forget to add the submodule before you commit.



Fig. 1 Git flow diagram

## Dealing with git submodules

**WARNING:** The underlying philosophy is that there should be a single development leader in charge of the interaction between pyrad and its public submodules so regular developers should not be concerned by this section.

### Installing a git submodule

The pyrad superproject contains a number of Open Source public libraries. In some of them, namely Py-ART, we wish to have an active collaboration and therefore we should be able to interact with the project using the git commands. This requires several steps. In the following we will describe them taking Py-ART as an example. For other products the steps would be analogous:

1. fork the project in the github.com repository (simply register as user and click fork in the main page in <https://github.com/ARM-DOE/pyart>). A copy of the master program will be created in your personal github space, i.e. <https://github.com/meteoswiss-mdr/pyart>)
2. In your local copy of Pyrad, from the directory where you want to keep the submodule add the submodule **from the forked version** of the library:

git submodule add [https://github.com/meteoswiss-mdr/pyart.git](https://github.com/jfigui/pyart.git)

A file .gitmodules will be created in the main directory of the Pyrad repository. This is a good point where to commit the submodule to the repository.

1. Create a new local branch of the forked version. This is going to be the branch where local developments will be made:

git checkout –b pyart-mch

1. Add the information of your working branch into your git config file:

git config –file=.gitmodules submodule.pyart.branch pyart-mch

This is another good point to commit to the repository.

### Updating the local submodule working branch with changes in the master public library

1. Place yourself in the superproject directory and change the information on url and branch contained in the .gitmodules file. Do not forget to synchronize everything:

git config –file=.gitmodules submodule.pyart.url <https://github.com/ARM-DOE/pyart.git>

git config –file=.gitmodules submodule.pyart.branch master

git submodule sync

Where [pyart] is the name of the submodule and the url is the url of the master public library.

1. Place yourself in the submodule directory, check that you are using the master branch and change branch otherwise and pull to update the local branch with the changes in the public library:

git branch

git checkout master

git pull

1. Syncronize the changes in the submodule with the superproject:

git submodule sync

1. Now your local master branch is updated with the additions of the main public library. You should commit these changes to your forked version in github. First place yourself in the main directory of the superproject and change back your url in your .gitmodules file:

git config –file=.gitmodules submodule.pyart.url [https://github.com/meteoswiss-mdr/pyart.git](https://github.com/jfigui/pyart.git)

1. As usual you have to sync the submodule:

git submodule sync

1. Finally you should push the changes to your fork by placing yourself in your submodule project and:

git push

1. Now change the working branch back to the regular pyart-mch:

git checkout pyart-mch

1. And place yourself in the superproject main to change the branch in the .gitmodules file back to your working branch:

git config –file=.gitmodules submodule.pyart.branch pyart-mch

1. Now to update your working branch simply place yourself in the submodule directory and merge the master with it:

git merge master

1. Solve any possible conflicts that arise and commit and push the result.
2. Place yourself in the superproject directory and commit all the changes

### Transferring changes from the local submodule working branch to the master public library

Ideally this should be the responsibility of a single person.

1. Place yourself in the local submodule directory and make sure you are using the master branch:

git branch

If you are not in the master branch change it:

git checkout master

1. Make all the changes you desire to make public in this local master branch.
   1. If you want to add a completely new file to the master from the pyart-mch branch you can use git checkout and the path to the new file, for example:

git checkout pyart-mch pyart/correct/noise.py

* 1. If the file already exists and you want to selectively apply some changes use:

git checkout –patch pyart-mch pyart/correct/noise.py

It will allow to interactively go through the differences between the files at each branch and apply the changes you desire.

1. Commit all the changes you have performed and push them to your forked public repository
2. In your forked public repository ([https://github.com/meteoswiss-mdr/pyart](https://github.com/jfigui/pyart)) select the branch master and click on “New pull request”. Select ARM-DOE:master as your target and make sure that meteoswiss-mdr:master is the origin. Once a pull request is open all new commits will be directly visible so there is no need to open a pull request for each new commit.

# References

1. <https://www.continuum.io/downloads>
2. <http://trmm-fc.gsfc.nasa.gov/trmm_gv/software/rsl/index.html>
3. <http://wradlib.org/>
4. <https://github.com/ARM-DOE/pyart>
5. <https://github.com/nguy/artview>
6. <https://github.com/nasa/DualPol>
7. <https://github.com/tjlang/SkewT>
8. <https://github.com/CSU-Radarmet/CSU_RadarTools>
9. <https://github.com/nasa/PyTDA>
10. <https://github.com/nasa/SingleDop>
11. <https://github.com/nasa/PyBlock>
12. <https://gitlab.meteoswiss.ch/MDR/pyrad>
13. <https://www.python.org/dev/peps/pep-0008/>
14. <https://pypi.python.org/pypi/pycodestyle>
15. <https://pycodestyle.readthedocs.io/en/latest/>