The bash scripts have been translated into a set of python scripts, and now also include the addition of a database handler to log success/failure of the steps in processing. This document explains how these scripts work.

There are 3 directories: convert, calc_calib and apply_calib.

1. convert
These scripts convert the vol, ele and azi raw binary files from the radar into netcdf cfradial files.
There are 3 scripts: convert_hour.py, convert_day.py and convert_time_series.py.

convert_hour is the basic unit of processing. This will process data on the command line, it won't send any jobs to lotus. It processes the files within each unit of time (each hour).

An example command to run this script is:

python convert_hour.py -t vol 2020040100

This would search for all the vol files within the hour 0000 to 0059 on 1st April 2020 and process them.

convert_day is the next layer up. This script will create batches of jobs and send them to lotus to run in parallel. The number of jobs per batch is determined by parameters in the SETTINGS file.

An example command to run this script is:

python convert_day.py -t vol -d 20200401

The default setting is for each batch to process 6 hours of data:

python convert_hour.py -t vol 2020040100 2020040101 2020040102 2020040103 2020040104 2020040105

convert_time_series is the top level where you can specify a range of dates you wish to process, which will be broken down into days and hours.

An example of the command to run this script is:

python convert_time_series.py -t vol -s 20200401 -e 20200430

this script will call convert_day for each day within the range
convert_day will then break each day down into batches of 6 hours data to send to lotus.

SETTINGS
There is a settings file for each stage of processing.
In this file you set the paths for where the raw data are located, where you want the processed data to go, the locations of the log files and also the RadxConvert parameters file.

Database Handler
The addition of the database handler is key to large batch processing. It enables the scripts to be re-run without having to process every file again. It creates a look-up table that labels files as being successfully processed or not. Only those that haven't been processed will be run subsequent times.
See additional document with a description of how to set this up and implement it on the Jasmin system.

2. calc_calib
This second stage of processing contains the bulk of the work, where the vertical scans are processed to calculate an offset for ZDR, the volume scans are used to calculate an offset for ZDR as a comparison to that derived from the vertical scans, and the volume scans are processed to calculate an offset for Z.

a. Vertical scans
There are three scripts:
process_vert_scans_day.py
process_vert_scans_time_series.py
process_hourly_zdr.py

For the first script it is called as follows:

python process_vert_scans_day.py -d 20200401 -p 0

where the argument p is a parameter to select whether plots are created or not. 0=no 1=yes.

For the database handler there are different success/failure identifiers.
The first check uses the weather station to see whether any rain fell on a given day. If there was less than 1mm, we move on to the next file and set an identifier of “no rain”.
If there was rain, when the file is processed, there might not be sufficient data to extract a vertical profile and make an estimate of ZDR, so the identifier “insufficient data” is applied if this is the case.

To process a time series we use the second script as follows:

python process_vert_scans_time_series.py -s 20200401 -e 20200430 -p 0

This script will call the day script and send each day as a separate job to lotus.

The first level of processing creates csv files containing values of ZDR bias and melting layer height for each radar file. The next step is to create averages over hourly periods. This is done using the following command:

python process_hourly_zdr.py

This script loops through all the days processed from the previous scripts and calculates hourly averages if sufficient data exists.

b. Volume scans for ZDR

We also use the volume scans to calculate an estimate of the bias in ZDR. This is more of a sanity check to verify what we find from the vertical scans. There is normally more data available from the volume scans.

Similar to the section a above, there are two scripts:

process_horz_zdr_day.py
python process_horz_zdr_day.py -d 20200401

process_horz_zdr_time_series.py
python process_horz_zdr_time_series.py -s 20200401 -e 20200430

There is currently no implementation of the database handler in these scripts

c. Volume scans for Z

There are two scripts for processing the volume scans to calculate an estimate of any bias in Z.

process_volume_scans_day.py
python process_volume_scans_day.py -d 20200401

This scripts performs the processing wherever it is run, i.e. on the local machine

The database handler is used again in this script to identify days that didnt have sufficient rain for processing, which could either be because the ZDR/ML file doesnt exist, or if it did, there weren't any suitable rays for doing the analysis.

process_volume_scans_time_series.py
python process_volume_scans_time_series.py -s 20200401 -e 20200430

The time series script sends each day to lotus as a separate job.

SETTINGS
Similarly to the convert stage, there is a SETTINGS file for specifying all the parameters and paths i.e. input/out directories for data, directory for output log files, location of ZDR files.

UTILITIES
There is a directory called utilities that contains two further python scripts.
calib_functions.py contains most of the routines used in the calibration process and these are called by the scripts described above.

See additional documents for description of calibration routines (RAINE ZDR Calibration Method v2, RAINE Z Calibration Method v2)

3. apply_calib
This step is fairly similar to step one, except that we are now applying the calibration offsets to the uncalibrated netcdf files.

There are two scripts:

apply_calc_chunk.py
python apply_calc_chunk.py -p 1 -f {} -t sur

where the value of ***p*** corresponds to which params file the user chooses. The params file sets the value of the offsets to be applied. ***f*** corresponds to the set of files to be processed.

apply_calc_time_series.py
python apply_calc_time_series.py -p 1 -s yyyymmddhhnnss -e yyyymmddhhnnss -t sur

Instead of sending each day to lotus, the script splits the total number of files into batches instead. This is because each params file doesn't necessarily start at the start of the day, it might apply half way through a day, not for the full day. We set a parameter for the size of the chunk of files we want (normally equivalent to 6 hours worth of data) and this batch of files is sent to lotus.