# I) Pool Location:

You can get the pool here:

[git@bear.cira.colostate.edu:rammb\_tc\_group/pool.git](mailto:git@bear.cira.colostate.edu:rammb_tc_group/pool.git)

The most recent version of the pool can be found on rammb-mirs2 in the following directory:

/home/jpss-tc/pool\_database\_processing/pool\_synced\_with\_master/pool

# II) Running the Pool

To run the pool, use the command:

python pool\_app COMMAND LIST\_OF\_CONFIG\_FILES

# III) Commands

The program has three commands:

create – Creates a pool database directory

update – Updates the pool database with new files

query – Generates a list of files within a bounding box and optionally generates a file containing every sample that is within the bounding box.

# IV) Command Examples

1) Create

To create an example nucaps pool run the command:

python pool\_app create tests/manual\_tests/database\_debug\_tests/simple\_nucaps\_database.cfg

This will create a directory "nucaps\_pool" in the tests/manual\_tests/database\_debug\_tests/database\_output directory.  If the "nucaps\_pool" directory already exists, then the program will fail.  A single file with meta data about the pool will be placed in the "nucaps\_pool" directory as well.

2)Update

To update the pool, run the command:

python pool\_app update tests/manual\_tests/database\_debug\_tests/simple\_nucaps\_database.cfg

This will search the directory specified in the "[file\_gather], gather\_dir" option for files matching the regular expression in the "[file\_gather], regex" option.  Each of these files will be added to the pool.  The "gather\_dir" in the above .cfg file is set to "tests/manual\_tests/database\_debug\_tests/debug\_file\_input/"

3)Query

To query the pool, run the command:

python pool\_app query tests/manual\_tests/database\_debug\_tests/simple\_nucaps\_database.cfg

This will query the pool using the options in the "[query]" section and write a list of files in the file specified by the "[query], output\_file" option.

# V) Config Cascade

The pool takes a list of config files as its second command line option. These config files will be read by the program and combined into one config in memory. The config files will be read in the order given. If an option appears in more than one config file, then the option contained in the last read config file will be used.

EX:

We have the command:

python pool\_app update foo.cfg bar.cfg

Where foo.cfg contains:

[some\_option\_section]

some\_option = foo.txt

And bar.cfg contains:

[some\_option\_section]

some\_option = bar.txt

Since the two config files contain the same option, but bar.cfg is read after foo.cfg, then the value for “some\_option” in memory will be “bar.txt”.

The config cascade is used to organize options so that options don’t need to be duplicated across files.

# VI)Query Files

A query command will always generate a list of files that intersected your bounding box. Due to the resolution of the grid used by the pool database, it is likely that this list may contain a few files that did not actually intersect your bounding box, but had a sample that was nearby.

A query command will optionally generate a query file. This file can be thought of as a collection of all the samples that intersected your bounding box. No extraneous samples will be included in this collection.

Once you’ve read a query file, you will have one array per variable. The first index in each of these arrays is the sample index.

EX:

Your query file contains a 1D Lat array, a 1D Lon array, and a 2D Temp\_Profile array. To access the lat, lon and temperature profile for sample 17, then you would use the following syntax:

lat = Lat[17]

lon = Lon[17]

profile = Temp\_Profile[17, :]

Query files are provided as either a HDF5 file or a .tar file containing ASCII files for each array. The ASCII files can be read using the cira\_txt\_reader module. The .tar files are compatible with “old” pool query reading code.