Explanation for contents in git/CMIP2\_CVs/src:

- AMIP\_full\_dict.py --- The code which produces the AMIP1&2\_source\_id.json file needed to run the reprocessing. When new information is found about models/aliases, it should be changed here and it will propagate through everything else. Things that were unknown at the time are marked as ‘???’ or ‘TBD’ in this script.

-big\_kahuna6\_1.py --- This is the workhorse code that needs to be run to set the data reprocessing in motion. More on this below.

CMIP2\_full\_dict.py --- Same thing as AMIP\_full\_dict.py but creates the CMIP1&2\_source\_id.json. Same things apply for unknowns and changes.

-GenerateXmlsForAMIP.py --- This code re-writes the xml files for all of the data under the AMIP2-STORAGE directory under PJG\_StorageRetreival. This was required because the old xmls were faulty. This was not necessary for the old CMIP xmls however. This code can be used to re-write all xmls under any sub-directory by changing the path. It only needs to be run once and produces xmls titled: “’variable’\_’alias’\_BEN.xml”

-kludgersTwo.py --- This code is a function needed to run big\_kahuna. This code handles the cmor\_write errors brought about by unit problems with input data. For example if the input data says ‘tas’ has units of ‘Pa’ or ‘ ‘, than it changes the units to ‘K’. So it handles unit exceptions for all variables. If there is a units error with cmor\_write, it needs to be added to this kludgersTwo.py script. Also, if a new variable is used, it must be added to this script. kludgersTwo can also carry out unit conversions, for example from mm/day to kg/s-m^2 for precip. These also must be added in each case it is discovered it is needed. The function also outputs messages concerning the variable units that are stored in the QC json file for each variable.

-list\_of\_model\_names\_from\_dirc.py --- This script takes a directory that holds old data and goes through to grab model information from netcdf files. For example, for each model in ‘tas’ it will produce a dictionary of models with a nested dictionary containing the lat, long, and level information for said model. Useful for comparing to the IPCC table and AMIP newsletters.

-Variable\_Table\_Matcher.py  --- This script is a function needed to run big\_kahuna. The function reads the path were all of the old data is located, extracts all the variables for which there is data, and then determines if they are in the Amon, Simon, Omon, or Lmon CMIP6.json tables (eg: CMIP6\_Amon.json). If the variable found in the path is not found in any of those four tables it is marked as “Missing.” The function then outputs a dictionary of the variables and the corresponding json table they are defined in. It also outputs a json file called “MissingVariables.json” with a list of the variables that were not matched. The output dictionary is then used by big\_kahuna to decide what table needs to be used with Cmor to reprocess the data, or if the variable must be skipped altogether, as is the case for the variables marked ‘missing’. If the variable\_id has changed for a variable in the time between CMIP 2 and CMIP6 (eg: from sit to sithick for sea ice thickness) than this conversion/matching can be done here on a case by case basis.

The code “big\_kahunaX\_X.py” is the code used to do all of the reprocessing for the old AMIP and CMIP data. To run this code properly it requires the two functions “kludgersTwo.py” and “Variable\_Table\_Matcher.py” to be run/(be recognized as functions) in the working directory of the users choice. The CMOR Tables (CMIP6\_Amon.json, CMIP6\_coordinates.json, etc…) files found in ~/git/CMIP2\_CVs/Tables and the source ID json files (CMIP1&2\_source\_id.json && AMIP1&2\_source\_id.json) found in ~git/CMIP2\_CVs must be moved to the users home directory in the directory structure homeDirc/git/CMIP2\_CVs/…. respectively.

Once the needed function files and json tables are in place the code can be run. What follows is a summary of each section of the code and what it does.

0. The user must input the paths (as DataPathinX) to be reprocessed and what type of data (CMIP or AMIP) it is (as DataTypeX) and make sure to add to the list variables ‘paths’ and ‘types’. This should be the only user input. The code then begins to loop through the list of path and breaks up further to account for the control and perturbed data in CMIP. The appropriate table paths, out paths, and save paths are then set based on the users home directory.

1. The variables in the path are then pushed through the Variable\_Table\_Matcher function. It outputs a list of variables to iterate through and a dictionary of variables and the json table they belong too. The variables are then looped through and ‘Vcount’ is the index for the variable in the list. If the Variable table matcher function has shown a variable to have nojson table match, it is skipped here and the next variable is processed.
2. Paths are set, the list of aliases for the variable is acquired a directory for the user\_input json files is created.
3. The user\_input.json files are created here. The source\_id.json files are used to fill in these as much as possible. If there is a match between the aliases in the directory and the model names taken from the IPCC report or AMIP reports, it is recorded in the source\_id.json and then transferred to the user input json file here. If there is no match, nonsense values are used to “fill in the blanks” to satisfy CMOR until a match can be found.
4. This section takes care of “Too hard” problems. Aka problems with the data that were encountered and thought to be too hard to solve right now. So the variables and aliases that cause these problems are recorded here and are skipped. The error message raised is transferred to the QC file as well.
5. The xmls for each alias are then acquired. If there is an old xml, it is used, as is the case with the CMIP data. For the AMIP data new xmls had to be generated and these new x\_X\_BEN.xml files are used. If no xml is found underneath an alias, then that alias is skipped. NOTE this happens for the ‘ensemble’ aliases in the AMIP data. This must be addressed in the future.
6. This section sets up CMOR. It collects the lat, lon, time, and level data (if it is a4D variable). The cmor axes are set by taking the units and bounds from the input data and using them as the table entries for each coordinate. There is a potential problem with the level data as it is set as plev19, somewhat arbitrarily. For the AMIP data this is even more concerning because it was not clear what the levels were based off of. Also, if the variable is 4D the cdm.setAutoBounds is turned on because of continuing errors.
7. Here the kludgers function is used. This function corrects units problems for each variable on a case by case basis. When a new problem is encountered (in cmor\_write) it must be added here. This function also carries out unit conversions which are also carried out on a case by case basis.
8. The outut cmor file is created here. Also for QC, averages and other data are recorded before the data is processed with cmor. If there is already an output cmor netcdf file in the outpath, it is erased to make room for the new cmor file.
9. Quality control is carried out here. The QC dictionary is recorded here. All of the data that ends up in the QC plots is also generated here.
10. This section produces the actual plots used for the QC. It adds titles, labels, legends ,etc.. It also removes extreme values from data (because missing data is often expressed as huge numbers) so that bounds are based off of real data and not the missing data.
11. Here the QC plots and QC dictionary.json for each variable are saved and sent to the outpath. The plots are saved as pdf. The code then exits the loop and goes on to the next variable in the list