This document describes the process of converting the GSOM monthly .csv file to the Common Data Model (CDM) file formatted tables. This process is run in the one command.

Monthly.csv to CDMLite and QC tables.

The code is written in Python version 3.7.9.

Step 1

Set the working directory and output directories

OUTDIR3= "where the header tables are to be saved"

OUTDIR2= "where the observations tables are to be saved"

OUTDIR = " where the CDMlite tables are to be saved "

os.chdir("where the .csv files are located")

Step 2

You have the option to use a list of required csv files to be converted , which is useful, for running multiple jobs at once by chunking all the csv files into txt lists of 5000.

my\_file = open("where your ls1.txt are located ", "r")

all\_filenames = my\_file.readlines()

for filename in all\_filenames:

df=pd.read\_csv(filename, sep=",")

print(all\_filenames)

Or to process all the files at once in one job use:

extension = 'csv'

all\_filenames = [i for i in glob.glob('\*.{}'.format(extension))]

for filename in all\_filenames:

df=pd.read\_csv(filename, sep=",")

print(all\_filenames)

Or to start the process from a specific last file use:

for filename in all\_filenames[all\_filenames.index('SWM00002338.qff'):] :

df=pd.read\_csv(filename, sep=",")

print(all\_filenames)

Step 3

The code reads the .csv file by only the columns of variables that a required for the CDMlite table.

usecols = ["STATION","LATITUDE","LONGITUDE","ELEVATION","DATE","NAME", "PRCP", "TMIN", "TMAX", "TAVG", "SNOW", "AWND"]

The code then proceeds to set up the columns for the daily CDMlite. The source flags are then converted to C3S source identifiers and the variables are converted to C3s variable identifiers. The code sets the value significance and decimal for each variable and the code also conducts any value conversions to CDM requirements such as temperature in degrees Kelvin and pressure to pascals etc. and sets the values to the required units and height of each the variables. The values that have been qc flagged by the GHCND test are flagged as failed.

The code reads an external .csv file to a df2 and adds data policy and record numbers information to the variable df by merging the two df on primary\_station\_id, this is conducted on each variable df separately. External .csv can be found at: {<https://github.com/glamod/glamod-nuim/tree/master/PYTHON_CDM_Conversion_code/external_csv_files_for_conversion_tables_code>}

The Code also sets up the qc table and adds the qc flags from the CDMlite df to a separate df (qct) for each variable. The original source flags are converted to C3S qc flags for the definition table.

Once this is completed the final CDMlite df and qc\_definition qct files are then saved as pipe separated files to the two separate out directories. The loop continues over all files until completed.

Daily.csv to CDMobs and CDMhead tables.

The code is written in Python version 3.7.9.

Step 1

Set the working directory and output directories

OUTDIR2= "where the header tables are to be saved"

OUTDIR = " where the observations tables are to be saved "

os.chdir("where the daily .csv files are located")

Step 2

You have the option to use a list of required csv files to be converted , which is useful, for running multiple jobs at once by chunking all the csv files into txt lists of 5000.

my\_file = open("where your ls1.txt are located ", "r")

all\_filenames = my\_file.readlines()

for filename in all\_filenames:

df=pd.read\_csv(filename, sep=",")

print(all\_filenames)

Or to process all the files at once in one job use:

extension = 'csv'

all\_filenames = [i for i in glob.glob('\*.{}'.format(extension))]

for filename in all\_filenames:

df=pd.read\_csv(filename, sep=",")

print(all\_filenames)

Or to start the process from a specific last file use:

for filename in all\_filenames[all\_filenames.index('SWM00002338.csv'):] :

df=pd.read\_csv(filename, sep=",")

print(all\_filenames)

Step 3

The code firstly creates the observations tables and qc definition tables from the csv files.

df.columns=["Station\_ID","Date","observed\_variable",observation\_value","quality\_flag","Measurement\_flag","Source\_flag","hour"]

The code then filters the required rows of variables.

df = df[df["observed\_variable"].isin(["SNWD", "PRCP", "TMIN", "TMAX", "TAVG", "SNOW", "AWND", "AWDR", "WESD"])]

The source flags are then converted to C3S source identifiers and the variables are converted to C3s variable identifiers. The code converts each of the variable values to CDM compliant values and ads the required CDM compliant elements relative to each variable such as value significance and units etc. The code converts the qc flags to either pass or fail. The code reads an external .csv file to a df2 and adds information such as data policy etc to each variable dfobs by merging the two df on primary\_station\_id, this is conducted on each variable df separately. External .csv can be found at: {<https://github.com/glamod/glamod-nuim/tree/master/PYTHON_CDM_Conversion_code/external_csv_files_for_conversion_tables_code>}

The code continues to create the header table from the observations table for each file.The code reads the observation df and only retains specific columns {cols\_list} into a df named merged\_df

{ col\_list = ["observation\_id", "report\_id", "longitude", "latitude", "source\_id","date\_time"]}

hdf=col\_list.copy()

The code then proceeds to set up the columns for the CDMhead {hdf} using some of the columns in the CDMobs {merged\_df}. The code reads an external .csv file to a df2 and adds required information to the variable hdf by merging the two df on primary\_station\_id. External .csv can be found at: {<https://github.com/glamod/glamod-nuim/tree/master/PYTHON_CDM_Conversion_code/external_csv_files_for_conversion_tables_code>}

The code then removes duplicate timestamp reports so that only one timestamp header report is now present.

hdf=hdf.drop\_duplicates(subset=['duplicates\_report'])

Once this is completed for all variables the CDMlite {df\_lite}, CDMhead {hdf} and CDMobs {df} are then saved as pipe separated files to the output directories. The loop continues over all files until completed.