e3sm_to_cmip

E3SM

Feb 09, 2023

FOR USERS

The `e3sm_to_cmip` package converts E3SM model output variables to the CMIP standard format. The tool supports variables in the atmospheric, land, MPAS ocean, and MPAS sea-ice realms. The handling of each realm is slightly different, so care must be made when dealing with the various data types (refer to *Preprocessing Data by Realm*)

CHAPTER

ONE

USAGE

First, follow the *Getting Started* page for how to access `e3sm_to_cmip` in an Anaconda environment.

Afterwards, there are two main ways to run `e3sm_to_cmip`:

1. Invoking the `e3sm_to_cmip` package directly on the appropriately pre-processed input files
   - *Usage Guide*
   - *Examples*
2. Using the automated CWL workflows provided in the `scripts/cwl_workflows` directory in the repository.
   - *Leveraging CWL Workflows*

1.1 Getting Started

1.1.1 Prerequisites

`e3sm_to_cmip` is distributed through conda, which is available through Anaconda and Miniconda. The instruction to install conda from Miniconda is provided as follows:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
```

Then follow the instructions for installation. To have conda added to your path you will need to type `yes` in response to "Do you wish the installer to initialize Miniconda3 by running conda init?" (we recommend that you do this). Note that this will modify your shell profile (e.g., `~/.bashrc`) to add `conda` to your path.

Note: After installation completes you may need to type `bash` to restart your shell (if you use bash). Alternatively, you can log out and log back in.

1.1.2 Installation

1. Create a conda environment from scratch with `e3sm_to_cmip` (conda create)

   We recommend using the Conda environment creation procedure to install `e3sm_to_cmip`. The advantage with following this approach is that Conda will attempt to resolve dependencies (e.g. `python >= 3.8`) for compatibility.

   To create an `e3sm_to_cmip` conda environment run:

   ```
   conda create -n <ENV_NAME> -c conda-forge e3sm_to_cmip
   conda activate <ENV_NAME>
   ```

2. Install `e3sm_to_cmip` in an existing conda environment (conda install)

   You can also install `e3sm_to_cmip` in an existing Conda environment, granted that Conda is able to resolve the compatible dependencies.

```
conda activate <ENV_NAME>
conda install -c conda-forge e3sm_to_cmip
```

3. Conda Development Environment and Source Code

   This environment is intended for developers.

   First, clone the repo and set up the conda dev environment:

```
git clone https://github.com/E3SM-Project/e3sm_to_cmip.git
cd e3sm_to_cmip
conda env create -f conda-env/dev.yml
conda activate e3sm_to_cmip_dev
```

   Once you have dev environment setup, simply run:

```
pip install .
```

1.1.3 Additional Dependencies

The following dependencies are required to run `e3sm_to_cmip` for all normal runs (`--info` or `--simple` flag not specified).

CMIP6 Controlled Vocabulary Tables

This repository needs to be cloned because it is not currently available as a data package on `conda-forge`.

1. Clone the repository

```
git clone https://github.com/PCMDI/cmip6-cmor-tables.git
```

2. Example usage with `e3sm_to_cmip`

```
 e3sm_to_cmip --help

-t <tables-path>, --tables-path <tables-path>
                    Path to directory containing CMOR Tables directory,
                    required unless the --simple flag is used.
```

CMIP6 Metadata Tables

This repository needs to be cloned because it is not currently available as a data package on `conda-forge`.

1. Clone the repository

```
git clone https://github.com/E3SM-Project/CMIP6-Metadata.git
```

2. Example usage with `e3sm_to_cmip`

```
 e3sm_to_cmip --help

-u <user_input_json_path>, --user-metadata <user_input_json_path>
                    Path to user json file for CMIP6 metadata, required
                    unless the --simple flag is used.
```

1.2 Preprocessing Data by Realm

The `e3sm_to_cmip` package can operate on variables in four realms: **atmosphere, land, ocean, and sea-ice**. Each realm requires different preprocessing steps for its related input datasets.

1.2.1 Atmosphere

Processing atmosphere variables requires that each of the input variables be provided in regridded time-series files (multiple files spanning different time segments is supported), and follow the NCO naming format of `VARNAME_START_END.nc` (e.g., `PRECC_185001_201412.nc`).

- NOTE: 3D atmosphere variables

A subset of the 3D atmosphere variables require that they be converted from the internal model vertical levels over to the `"plev19"` levels. A copy of the vertical remap file can be found here

CMIP6 files requiring this vertical level change are: `"hur", "hus", "ta", "ua", "va", "wap", and "zg"`.

1.2.2 Land

Similarly to the atmospheric variables, land variables must be provided as regridded time-series files in the NCO naming format.

1.2.3 MPAS ocean

MPAS Ocean variables are regridded at run time, so they don't require preprocessing. These files should all be present in the input directory. Data required for ocean processing are as follows:

1. `mpaso.hist.am.timeSeriesStatsMonthly`
2. mpaso_in or mpas-o_in
3. one mpaso restart file

1.2.4 MPAS sea-ice

MPAS sea-ice variables are regridded at run time, and they don't preprocessing. These files should all be present in the input directory. Data required for ocean processing are as follows:

1. `mpascice.hist.am.timeSeriesStatsMonthly` or `mpassi.hist.am.timeSeriesStatsMonthly`
2. mpassi_in or mpas-cice_in
3. one mpassi or mpascice restart file

1.3 Usage

The code block below shows the available flags when running `e3sm_to_cmip`. Please be aware that some arguments are required or optional based on how `e3sm_to_cmip`.

```
$ e3sm_to_cmip --help
usage: e3sm_to_cmip [-h]
Convert ESM model output into CMIP compatible format
required arguments (general):
-v  [ ...], --var-list  [ ...]
                        Space separated list of variables to convert from
                        E3SM to CMIP.
required arguments (without --info):
-i , --input-path     Path to directory containing e3sm time series data
                        files. Additionally namelist, restart, and 'region'
                        files if handling MPAS data. Region files are
                        available from https://web.lcrc.anl.gov/public/e3sm/i
                        nputdata/ocn/mpas-o/<mpas_mesh_name>.
required arguments (without --simple):
-o , --output-path    Where to store cmorized output.
-t <tables-path>, --tables-path <tables-path>
                        Path to directory containing CMOR Tables directory,
                        required unless the `--simple` flag is used.
-u <user_input_json_path>, --user-metadata <user_input_json_path>
                        Path to user json file for CMIP6 metadata, required
                        unless the `--simple` flag is used.
required arguments (with --realm [mpasso|mpassi]):
--map <map_mpas_to_std_grid>
                        The path to an mpas remapping file. Required if realm
                        is 'mpaso' or 'mpassi'. Available from
                        https://web.lcrc.anl.gov/public/e3sm/mapping/maps/.
optional arguments (general):
-n <nproc>, --num-proc <nproc>
                        Optional: number of processes, default = 6. Not used
                        when -s, --serial specified.
--debug               Set output level to debug.
--timeout TIMEOUT     Exit with code -1 if execution time exceeds given
                        time in seconds.
-H <handler_path>, --handlers <handler_path>
                        Path to cmor handlers directory, default is the
                        (built-in) 'e3sm_to_cmip/cmor_handlers'.
--precheck PRECHECK   Check for each variable if it's already in the output
                        CMIP6 directory, only run variables that don't have
                        pre-existing CMIP6 output.
--logdir LOGDIR       Where to put the logging output from CMOR.
--custom-metadata CUSTOM_METADATA
                        The path to a json file with additional custom
                        metadata to add to the output files.
optional arguments (run mode):
--info                Print information about the variables passed in the
                        --var-list argument and exit without doing any
                        processing. There are three modes for getting the
                        info, if you just pass the --info flag with the
```

(continues on next page)

(continued from previous page)

```
                        --var-list then it will print out the information for
                        the requested variable. If the --freq <frequency> is
                        passed along with the --tables-path, then the CMIP6
                        tables will get checked to see if the requested
                        variables are present in the CMIP6 table matching the
                        freq. If the --freq <freq> is passed with the
                        --tables-path, and the --input-path, and the input-
                        path points to raw unprocessed E3SM files, then an
                        additional check will me made for if the required raw
                        variables are present in the E3SM output.
--simple              Perform a simple translation of the E3SM output to
                        CMIP format, but without the CMIP6 metadata checks.
                        (WARNING: NOT WORKING AS OF 1.8.2)
-s, --serial          Run in serial mode (by default parallel). Useful for
                        debugging purposes.
optional arguments (run settings):
--realm <realm>       The realm to process. Must be atm, lnd, mpaso or
                        mpassi. Default is atm.
-f FREQ, --freq FREQ  The frequency of the data (default is 'mon' for
                        monthly). Accepted values are 'mon', 'day', '6hrLev',
                        '6hrPlev', '6hrPlevPt', '3hr', '1hr.
optional arguments (with --info):
--info-out INFO_OUT   If passed with the --info flag, will cause the
                        variable info to be written out to the specified file
                        path as yaml.
helper arguments:
-h, --help            show this help message and exit
--version             Print the version number and exit.
```

Additional descriptions of some of the arguments can be found below.

1.3.1 Required arguments (general):

Variable List

The "–var-list" or "-v" flag is a mandatory option, and should be a list of CMIP6 variable names to be output.

1.3.2 Required arguments (without –info)

Input Path

This mandatory flag should point at a directory containing the data files to be processed.

User Input Metadata

The "–user-metadata" or "-u" flag should be the path to a json formatted metadata file containing CMIP6 metadata for the case being processed. This flag can be avoided for non-official data by using the "–simple" flag. Otherwise, the file should look something like the metadata files that can be found here

Tables Path

The "–tables-path" or "-t" flag should point to the "Tables" directory of the CMIP6 controlled vocabulary repository. The repository can be found here

1.3.3 Required arguments (without –simple)

Output Path

This mandatory flag is the location that all output files will be placed. The main output is a directory named CMIP6, which contains the CMIP6 directory structure, with the output files as leaf nodes. Other output files include a copy of the user metadata (if present), and a directory named cmor_logs containing the per-variable log files generated by CMOR.

1.3.4 Required arguments (with –realm [mpasso|mpassi])

MPAS mapfile

When processing MPAS ocean or sea-ice variables, a mapfile is needed for regridding. Use the "–map" flag to pass the path to this mapfile.

1.3.5 Optional arguments (general)

Numproc

By default, the variable converters are run in parallel using a process pool with 6 worker processes. The "–num-proc" or "-n" flag can be used to control the number of simultaneously executing processes. For example, 3D ocean fields take significantly more RAM then other variables, so the number of converters running at once may be reduced to accommodate the machine being used.

Handler Path

A directory of custom variable handlers can be passed using the "–handlers" or "-H" flag.

Custom Metadata

Additional custom metadata can be added to the global attributes of the output files by using the "–custom-metadata" flag to point to a json formatted file containing the metadata key value pairs.

1.3.6 Optional arguments (run mode)

Info

The "–info" flag can be used in three different ways to determine information about the variables being requested for processing. In the simplest form, passing only the "–info" and "–var-list" flags will return information about the required input and CMIP6 output names of the variables passed in the variable list.

If the –freq <frequency> is passed along with the –tables-path, then the CMIP6 tables will get checked to see if the requested variables are present in the CMIP6 table matching the freq.

If the –freq <freq> is passed with the –tables-path, and the –input-path, and the input-path points to raw unprocessed E3SM files, then an additional check will me made for if the required raw variables are present in the E3SM output. In this last mode, instead of passing a directory of time-series files as the input path, pass the path to raw unprocessed E3SM cam or eam files.

Simple

This optional flag will cause the tool to run without needing or checking for the custom CMIP metadata usually required for processing. Output from this mode use the same converter code as the default mode, but the output doesnt contain the required metadata needed for a CMIP publication. This mode should be used when the output is intended for analysis, but is not suited for publication.

Serial

For debugging purposes, or when running in a resource constrained environment, the "–serial" or "-s" boolean flag can be used to cause the conversion process to be run in serial, using the main process.

1.3.7 Optional arguments (run settings)

Realm

The type of realm being operated on should be specified using the "–realm" flag. Allowed values are "atm", "lnd", "mpaso" and "mpassi." This is needed so that the package can correctly determine what type of input files to look for. By default "atm".

Frequency

The "–freq" and "-f" flags can be used to process high-frequency datasets. By default the tool assumes its working with monthly data. The following submonthly frequencies are supported: [6hr, 6hrLev, 6hrPlev, 3hr, day]

1.4 Examples

1.4.1 Simple atmosphere variable example

The first step in converting atmosphere variables is to do the regridding and time-series extraction. For this example, assume there's a directory named "atmos-input" that contains a single year of cam.h0 monthly history files, and the target is to produce the "pr," and "clt" variables.

The first step is to run a query to find what source variables are needed for these two output variables:

```
>> e3sm_to_cmip --info -v pr, clt
[*]
CMIP6 Name: clt,
CMIP6 Table: CMIP6_Amon.json,
CMIP6 Units: %,
E3SM Variables: CLDTOT,
Unit conversion: 1-to-%
[*]
CMIP6 Name: pr,
CMIP6 Table: CMIP6_Amon.json,
CMIP6 Units: kg m-2 s-1,
E3SM Variables: PRECC, PRECL
```

This shows that the clt CMIP variable needs the raw CLDTOT input variable, and pr needs both PRECC and PRECL.

The next step is to use ncclimo to extract the time-series and do the regridding. A detailed tutorial can be found here

This example uses files which look like "20191204.BDRD_CNPCTC_SSP585_OIBGC.ne30_oECv3.compy.cam.h0.1850-01.nc", and so their casename is "20191204.BDRD_CNPCTC_SSP585_OIBGC.ne30_oECv3.compy".

```
>> ncclimo --start=0001 --end=0001 --ypf=1 -c <CASENAME> -o ./native -O ./regrid -v␣
↪CLDTOT,PRECC,PRECL -i ./atmos-input --map=<PATH TO YOUR MAPFILE>
Climatology operations invoked with command:
/export/baldwin32/anaconda3/envs/cwl/bin/ncclimo --start=2015 --end=2015 --ypf=1 -c␣
↪20191204.BDRD_CNPCTC_SSP585_OIBGC.ne30_oECv3.compy -o ./native --regrid=./regrid -v␣
↪CLDTOT,PRECC,PRECL -i ./atm_testing -v CLDTOT,PRECC,PRECL --map=/export/zender1/data/
↪maps/map_ne30np4_to_cmip6_180x360_aave.20181001.nc
Started climatology generation for dataset 20191204.BDRD_CNPCTC_SSP585_OIBGC.ne30_oECv3.
↪compy at Tue Nov 24 15:05:20 PST 2020
Running climatology script ncclimo from directory /export/baldwin32/anaconda3/envs/cwl/
↪bin
NCO binaries version 4.9.3 from directory /export/baldwin32/anaconda3/envs/cwl/bin
Parallelism mode = background
Timeseries will be created for each of 3 variables
Background parallelism processing variables in var_nbr/job_nbr = 3/3 = 1 sequential␣
↪batches of job_nbr = 3 simultaneous jobs (1 per variable), then remaining 0 jobs/
↪variables simultaneously
Will split data for each variable into one timeseries of length 1 years
Splitting climatology from 12 raw input files in directory ./atm_testing
Each input file assumed to contain mean of one month
Native-grid split files to directory ./native
Regridded split files to directory ./regrid
Tue Nov 24 15:05:21 PST 2020: Generated ./native/CLDTOT_201501_201512.nc
Tue Nov 24 15:05:21 PST 2020: Generated ./native/PRECC_201501_201512.nc
```

(continues on next page)

(continued from previous page)

```
Tue Nov 24 15:05:21 PST 2020: Generated ./native/PRECL_201501_201512.nc
Tue Nov 24 15:05:22 PST 2020: Regridded ./regrid/CLDTOT_201501_201512.nc
Tue Nov 24 15:05:22 PST 2020: Regridded ./regrid/PRECC_201501_201512.nc
Tue Nov 24 15:05:22 PST 2020: Regridded ./regrid/PRECL_201501_201512.nc
Quick plots of last variable split in last segment:
ncview ./regrid/PRECL_201501_201512.nc &
panoply ./regrid/PRECL_201501_201512.nc &
Completed 1-year climatology operations for dataset with caseid = 20191204.BDRD_CNPCTC_
↪SSP585_OIBGC.ne30_oECv3.compy at Tue Nov 24 15:05:22 PST 2020
Elapsed time 0m2s
```

The next step is to call the e3sm_to_cmip package and use the time-series files as input:

```
>> e3sm_to_cmip -i regrid/ -o cmip_output -v prc, clt -t <PATH TO CMIP6 TABLES> -u <PATH␣
↪TO CMOR USER INPUT JSON>
[*] Writing log output to: cmip_output/converter.log
[+] Running CMOR handlers in parallel
100%|| 2/2 [00:01<00:00,  1.96it/s]
[+] 2 of 2 handlers complete
```

Alternately, if the data isn't going to be published to CMIP6, the "simple" mode can be used which doesnt require the full CMIP6 tables or metadata and produces output files that are very close to the CMIP6 requirements, but with placeholder metadata

```
>> python -m e3sm_to_cmip -i regrid -o cmip_output -v prc, clt --simple
[*] Writing log output to: cmip_output/converter.log
[+] Running CMOR handlers in parallel
[+] writing out variable to file /cmip_output/prc_CMIP6_Amon_201501-201512.nc
↪                                                                              ␣
↪                                                                              ␣
↪                                                                              ␣
↪                                                  | 0/2 [00:00<?, ?it/s][+] writing␣
↪out variable to file /p/user_pub/e3sm/baldwin32/workshop/ssp585/ssp585/output/pp/cmor/
↪ssp585/2015_2100/cmip_output/prc_CMIP6_Amon_201501-201512.nc
[+] writing out variable to file /cmip_output/clt_CMIP6_Amon_201501-201512.nc
100%|| 2/2 [00:00<00:00,  6.79it/s]
[+] 2 of 2 handlers complete
```

1.4.2 Plev atmosphere variable example

Some 3D atmosphere CMIP6 variables are on the plev19 vertical levels instead of the model and require remapping from the default model levels to the plev19 levels. These variables can be distinguished from model-level variables by the `Levels` field in their info having the name `plev19`.

An example us the `hus` variable

```
>> e3sm_to_cmip --info -v hus
[*]
CMIP6 Name: hus,
CMIP6 Table: CMIP6_Amon.json,
CMIP6 Units: 1,
E3SM Variables: Q
Levels: {'name': 'plev19', 'units': 'Pa', 'e3sm_axis_name': 'plev'}
```

Before performing the horizontal remapping, the raw files must first be vertically remapped using the following command and the plev19 vertical remapping file which can be found here

```
mkdir vrt_regrid
for file in `ls atm-input`
do
  ncks --rgr xtr_mth=mss_val --vrt_fl=vrt_remap_plev19.nc ./atm-input/$file ./vrt_regrid/
↪$file
done
```

The output files will be converted from the default 72 vertical levels which come out of the E3SM model into 19 vertical levels defined by the CMIP6 project. These files can then be regridded and converted as in the example above.

1.4.3 End-to-End High Frequency Example

The first step is to check what variables in the raw input data are possible to be converted at the desired frequency. For this we need to use the "info" option and give it three things, the frequency of data we want to convert, the input path to the raw data (not time-series, but native model output), and the location of our copy of the CMIP6 controlled vocabulary tables:

```
>> e3sm_to_cmip --info -v all --input /p/user_pub/work/E3SM/1_0/historical/1deg_atm_60-
↪30km_ocean/atmos/native/model-output/day/ens1/v1/ --tables ~/projects/cmip6-cmor-
↪tables/Tables/
[*]
CMIP6 Name: huss,
CMIP6 Table: CMIP6_day.json,
CMIP6 Units: 1,
E3SM Variables: QREFHT
[*]
CMIP6 Name: tas,
CMIP6 Table: CMIP6_day.json,
CMIP6 Units: K,
E3SM Variables: TREFHT
[*]
CMIP6 Name: tasmin,
CMIP6 Table: CMIP6_day.json,
CMIP6 Units: K,
E3SM Variables: TREFHTMN
[*]
CMIP6 Name: tasmax,
CMIP6 Table: CMIP6_day.json,
CMIP6 Units: K,
E3SM Variables: TREFHTMX
[*]
CMIP6 Name: rlut,
CMIP6 Table: CMIP6_day.json,
CMIP6 Units: W m-2,
E3SM Variables: FLUT
```

The next step is to find and setup the corresponding CWL workflow, in this case since we're processing daily data we want to use the "atm-day" workflow under e3sm_to_cmip/scripts/cwl_workflows which you can find here. The CWL parameter file atm-day-job.yaml needs to be edited with the values for our case. We need to take the E3SM variable names given by the "–info" request earler and put them into the `std_var_list` parameter, and take the CMIP6 variable

names and put them into the std_cmor_list parameter. Create a new directory to hold your output, and place the new parameter file there.

```
# path to the raw model data
data_path: /p/user_pub/work/E3SM/1_0/historical/1deg_atm_60-30km_ocean/atmos/native/
↪model-output/day/ens1/v1/

# size of output data files in years
frequency: 25

# number of ncremap workers
num_workers: 12

# slurm account info
account: e3sm
partition: debug
timeout: 2:00:00

# horizontal regridding file path
hrz_atm_map_path: /export/zender1/data/maps/map_ne30np4_to_cmip6_180x360_aave.20181001.nc

# path to CMIP6 tables directory
tables_path: /export/baldwin32/projects/cmip6-cmor-tables/Tables/

# path to CMOR case metadata
metadata_path: /p/user_pub/e3sm/baldwin32/resources/CMIP6-Metadata/1.0/historical_ens1.
↪json

# list if E3SM raw variable names
std_var_list: [QREFHT, TREFHT, TREFHTMN, TREFHTMX, FLUT]

# list of CMIP6 variable names
std_cmor_list: [huss, tas, tasmin, tasmax, rlut]
```

Make a temp directory to contain the intermediate files created by the workflow, and set it as your TMPDIR

```
cd /p/user_pub/e3sm/baldwin32/workshop/highfreq/1.0/historical
mkdir tmp
export TMPDIR=/p/user_pub/e3sm/baldwin32/workshop/highfreq/1.0/historical/tmp
```

And startup the CWL workflow

```
>> cwltool --tmpdir-prefix=$TMPDIR --preserve-environment UDUNITS2_XML_PATH ~/projects/
↪e3sm_to_cmip/scripts/cwl_workflows/atm-day/atm-day.cwl historical-atm-day-ens1.yaml
```

This will launch a fairly long running job as it steps through all the parts of the workflow. If you're running a very large set of data, it can help to use the nohup tool to wrap the command so it doesnt get interupted by logging out.

1.4.4 Simple MPAS Ocean variable example

Unlike Atmos and Land data, e3sm_to_cmip can work directly with the native MPAS Ocean (and Sea-Ice) model output files to cmorize selected variables.

The command line requires the following inputs (example for variable "thetao"):

```
--realm mpaso
-v thetao
--input The path to your input directory. [Raw MPAS ocean datafiles, plus namelist,␣
↪restart, and mappings files[*]]
--map The path to an mpas remapping file. [Required for realm mpaso and mpassi. ␣
↪Available from https://web.lcrc.anl.gov/public/e3sm/mapping/maps/
--user-metadata <path_to_your_metadata/name.json> [Required unless "-simple" is␣
↪specified]
--tables-path <Path to directory containing CMOR Tables directory> [Required unless
↪"-simple" is specified]
--output-path <Path to the directory for generated output>
```

[*] The input directory for MPAS processing must also include

```
namelist:  mpaso_in
restart:   (e.g.) mpaso.rst.1851-01-01_00000.nc (from the native output)
regionfile: (e.g.) oEC60to30v3_Atlantic_region_and_southern_transect.nc
```

Region files are available from https://web.lcrc.anl.gov/public/e3sm/inputdata/ocn/mpas-o/<mpas_mesh_name>.

1.5 CWL Workflows

There is a set of CWL workflow scripts in the repository (`/scripts/cwl_workflows`) for each realm. Each workflow breaks the input files up into manageable segment size and perform all the required input processing needed before invoking `e3sm_to_cmip`. These scripts have been designed to run on a SLURM cluster in parallel and will process an arbitrarily large set of simulation data in whatever chunk size required.

1.5.1 Setting up your CWL environment

To use the CWL workflows you will need additional dependencies in your environment:

```
conda install -c conda-forge cwltool nodejs
```

When CWL runs it needs somewhere to store its intermediate files. By default it will use the systems $TMPDIR but in some cases that wont work, for example on NERSC the compute nodes wont have access to the login nodes /tmp directory. An easy solution for this is to create a directory on a shared mount, and run `export TMPDIR=/path/to/shared/location` and then when running the cwltool use the `--tmpdir-prefix=$TMPDIR` argument.

1.5.2 Using the CWL Workflows

Each of the directories under `scripts/cwl_workflows` holds a single self-contained workflow. The name of the workflow matches the name of the directory, for example under the mpaso directory is a file named `mpaso.cwl` which contains the workflow.

The beginning of each workflow contains an `inputs` section which defines the required parameters, for example

```
inputs:
    data_path: string
    metadata: File
    workflow_output: string

    mapfile: File
    frequency: int

    namelist_path: string
    region_path: string
    restart_path: string

    tables_path: string
    cmor_var_list: string[]

    timeout: int
    partition: string
    account: string
```

Along with each of the cwl workflows is an example yaml parameter file, for example along with `mpaso.cwl` is `mpaso-job.yaml` which contains the following:

```
data_path: /p/user_pub/e3sm/staging/prepub/1_1_ECA/ssp585-BDRD//1deg_atm_60-30km_ocean/
↪ocean/native/model-output/mon/ens1/v0/
workflow_output: /p/user_pub/e3sm/baldwin32/workshop/ssp585/ssp585/output/pp/cmor/ssp585/
↪2015_2100

metadata:
    class: File
    path: /p/user_pub/e3sm/baldwin32/workshop/ssp585/ssp585/output/pp/cmor/ssp585/2015_
↪2100/user_metadata.json
mapfile:
    class: File
    path: /export/zender1/data/maps/map_oEC60to30v3_to_cmip6_180x360_aave.20181001.nc

frequency: 5
namelist_path: /p/user_pub/e3sm/baldwin32/workshop/E3SM-1-1-ECA.hist-bgc/mpaso_in
region_path: /p/user_pub/e3sm/baldwin32/resources/oEC60to30v3_Atlantic_region_and_
↪southern_transect.nc
restart_path: /p/user_pub/e3sm/baldwin32/workshop/E3SM-1-1-ECA.hist-bgc/mpaso.rst.1851-
↪01-01_00000.nc
tables_path: /export/baldwin32/projects/cmor/Tables

timeout: 10:00:00
account: e3sm
partition: debug
```

(continues on next page)

(continued from previous page)

```
cmor_var_list: [masso, volo, thetaoga, tosga, soga, sosga, zos, masscello, tos, tob, sos,
↪ sob, mlotst, fsitherm, wfo, sfdsi, hfds, tauuo, tauvo, thetao, so, uo, vo, wo,␣
↪hfsifrazil, zhalfo]
```

Once the parameter file is complete, the workflow can be executed by calling the cwltool

```
cwltool --tmpdir-prefix=$TMPDIR ~/projects/e3sm_to_cmip/scripts/cwl_workflows/mpaso/
↪mpaso.cwl mpaso-job.yaml
```

1.6 Variable Handlers

1.6.1 What are Variable Handlers?

In `e3sm_to_cmip`, each supported CMIP6 variable has a CMOR "handler". A handler defines the metadata necessary for CMORizing E3SM variable(s) to their equivalent CMIP6 variable.

1.6.2 How are Variable Handlers Defined?

The metadata for variable handlers are defined in (key, value) pairs.

Required metadata:

Key	Type	Example	Description
name	string	tas	CMIP6 variable name
units	string	K	CMIP6 variable's units
raw_variables	array of string(s)	[TREFHT]	The E3SM variable name(s) used in the conversion to the CMIP6 variable.
table	string	CMIP6_Amon.json	The default CMOR table filename. (Source: https://github.com/PCMDI/cmip6-cmor-tables/Tables)

Optional metadata (based on variable, default is `null`):

Key	Type	Example	Description
unit_conversion	string	ug-to-kg	An optional unit conversion formula for the final output data.
formula	string	tas	An optional conversion formula for calculating the final output data. Usually this is defined if there are more than one raw variable.
positive	"down" or "up"	down	The "positive" directive to CMOR enables data providers to specify the direction that they have assumed in fields (i.e. radiation fluxes has up or down direction) passed to CMOR. If their direction is opposite that required by CMIP6 (as specified in the CMOR tables), then CMOR will multiply the field by -1, reversing the sign for consistency with the data request.
levels	dictionary	{name: plev19, units: Pa, e3sm_axis_name: plev}	Distinguishes model-level variables, which require remapping from the default model level to the level defined in the levels dictionary.

1.6.3 Where are Variable Handlers Stored?

The supported CMIP6 variables and their handlers are defined in the E3SM CMIP6 Data Conversion Tables Confluence page.

These handler definitions are transferred to the `e3sm_to_cmip` repository at the following locations:

1. `e3sm_to_cmip/cmor_handlers/handlers.yaml`
    - **Stores handler definitions for atmosphere and land variables.**
    - Each top-level entry defines a handler.
    - There can be more than one handler per variable (e.g., to handle different frequencies)
    - Example:

```yaml
-   name: pr
    units: kg m-2 s-1
    raw_variables: [PRECT]
    table: CMIP6_day.json
    unit_conversion: null
    formula: PRECT * 1000.0
    positive: null
    levels: null
-   name: pr
    units: kg m-2 s-1
    raw_variables: [PRECC, PRECL]
    table: CMIP6_Amon.json
    unit_conversion: null
    formula: (PRECC + PRECL) * 1000.0
    positive: null
    levels: null
```

2. **`e3sm_to_cmip/cmor_handlers/vars` directory (*legacy design*)**
    - Each handler is defined as Python module (`.py` file).
    - Stores **legacy atmosphere land variable handlers** including: `areacella.py`, `clisccp.py`, `clcalipso.py`, `orog.py`, `pfull.py`, and `phalf.py`.
    - These handlers will be refactored into `handlers.yaml`. They either depend on CDAT modules or contain replicated code that has since been generalized.
    - **Please avoid defining new handlers in this directory. Instead, add new handlers to** `handlers.yaml`.
3. **`e3sm_to_cmip/cmor_handlers/mpas_vars` directory**
    - Each handler is defined as Python module (`.py` file).
    - Stores **handler definitions for MPAS ocean and sea-ice variable handlers.**
    - MPAS variables require additional processing requirements (e.g., use of mesh files).
    - The development team is considering refactoring the design of these handlers.

1.6.4 Working with Atmosphere and Land Variables

Realms: `atm`, `lnd`

How to add atmosphere/land handlers

1. Append a new entry to the `e3sm_to_cmip/cmor_handlers/handlers.yaml` file.
2. If the handler has a `formula`, add a formula function to `e3sm_to_cmip/cmor_handlers/_formulas.py`.

- The function parameters must include `data` (a dictionary mapping variables to its underlying `np.ndarray`) and `index` (the index within the array to apply the formula to).
- Example:

```
def cLitter(data: Dict[str, np.ndarray], index: int) -> np.ndarray:
    """
    cLitter = (TOTLITC + CWDC)/1000.0
    """
    outdata = (data["TOTLITC"][index, :] + data["CWDC"][index, :]) / 1000.0

    return outdata
```

How `e3sm_to_cmip` derives atmosphere/land handlers

`e3sm_to_cmip` **derives the appropriate variable handlers to use based on the available E3SM variables in the input datasets.** Afterwards, it applies any necessary unit conversions, formulas, etc. during the CMORizing process.

For example, let's say we want to CMORize the variable `"pr"` and we pass an E3SM input dataset that has the variables `"PRECC"` and `"PRECL"`. `e3sm_to_cmip` derives the appropriate `"pr"` variable handler using this logic flow:

1. Run `e3sm_to_cmip --var-list pr --input-path <SOME_INPUT_PATH>`
2. In `e3sm_to_cmip`, `--var-list` is stored in a Python list (`var_list=["pr"]`).
3. All defined handlers are gathered in a dictionary called `available_handlers`:

```
# Key = CMIP variable id,  value = list of available handler objects␣
↪defined in ``handlers.yaml`` and `/cmor_handlers`
available_handlers = {
        "pr": [
            VarHandler(name="pr", raw_variables=["PRECT"]),
            VarHandler(name="pr", raw_variables=["PRECC", "PRECL"]),
        ],
    }
```

4. Loop over `var_list`:
   a. Get the list of handlers from `available_handlers` dict (for `"pr"`)

```
[
    VarHandler(name="pr", raw_variables=["PRECT"]),
    VarHandler(name="pr", raw_variables=["PRECC", "PRECL"]),
]
```

   b. Derive a handler using the variables in the E3SM input dataset

- The E3SM input dataset contains "PRECC" and "PRECL", so we derive the second handler, `VarHandler(name="pr", raw_variables=["PRECC", "PRECL"])`.
- If no handler can be derived, an error is raised.

c. Append derived handler to final list of `derived_handlers`

5. Return `derived_handlers=[VarHandler(name="pr", raw_variables=["PRECC", "PRECL"])]`

1.6.5 Working with MPAS Ocean and Sea-ice Variables

Realms: `mpaso`, `mpassi`, `SImon`, and `Omon`

How to add MPAS variable handlers

Adding a variable handler for MPAS variable is slightly more involved process than for an atmosphere/land variable.

You need to create a Python module in `/cmor_handlers/mpas_vars`. We recommend taking a look at the existing modules such as `so.py` to get idea on how to add an MPAS handler.

How `e3sm_to_cmip` derives MPAS variable handlers

MPAS variable handlers are derived differently than atmosphere/land variables. Instead of deriving handlers by checking if the raw E3SM variable keys are found in the input E3SM datasets, **MPAS variable handlers use extra input files.**

Within an MPAS variable handler module, a `RAW_VARIABLES` static variable is instantiated (e.g., `RAW_VARIABLES = ["MPASO", "MPAS_mesh", "MPAS_map"]`).

- `"MPASO"` - The time series files
- `"MPAS_MESH"` - The mesh file
- `"MPAS_map"` - The mapping file

These individual elements are input files and not actually raw variables found in a dataset. All of these files are used to convert the E3SM variable to the CMIP variable.

1.7 High Frequency Data

Data from sub-monthly frequency files (i.e. high frequency data) can be processed the same way as monthly data, however the ncclimo commands to extract the time-series files is slightly different. Here's an example of extracting high-frequency time-series:

```
in_dir=./raw_data_path/
out_dir=./regridded_time_series/
native_out=./native_grid_time_series/
flags='-7 --dfl_lvl=1 --no_cll_msr --clm_md=hfs'
variables='PRECT TS'
start_year='1850'
end_year='2000'
years_per_output_file='50'
mapfile=${DATA}/maps/map_ne30v3_to_cmip6_180x360_aave.nc
ncclimo ${flags} -v ${variables} -O ${out_dir} -o {native_out} --map=${mapfile} -c␣
↪historical -i ${in_dir}
```

(continues on next page)

(continued from previous page)

Once you've extracted the time-series files, simply use the path to where they're stored as the input path argument to e3sm_to_cmip, and supply the `--freq` flag with the appropriate frequency.

1.8 Development Guide

1.8.1 Testing changes and debugging in Python

In older versions of `e3sm_to_cmip`, the only way to test changes was to run `e3sm_to_cmip` directly on the command line. This debugging process often involves adding `print` or `ipdb` statements throughout the codebase, which generally isn't good practice because it is inefficient and developers might forget to delete those statements.

As of `e3sm_to_cmip > 1.91`, `e3sm_to_cmip` can now be executed through a Python script. Advantages of using this approach include:

- Testing and debugging changes are significantly more efficient, which shortens the debugging cycle.
- Leverage IDEs to set breakpoints in IDEs and step through the call stack at runtime.
- Gain a better sense of how functions are manipulating variables and whether the correct behaviors are being produced.
- Prototype code in the debugger and implement those changes, then test if it behaves as expected.

Example 1 (CMORizing serially)

CLI Execution

```
e3sm_to_cmip --output-path ../qa/tmp --var-list 'pfull, phalf, tas, ts, psl, ps, sfcWind,
↪ huss, pr, prc, prsn, evspsbl, tauu, tauv, hfls, clt, rlds, rlus, rsds, rsus, hfss, cl,
↪ clw, cli, clivi, clwvi, prw, rldscs, rlut, rlutcs, rsdt, rsuscs, rsut, rsutcs, rtmt,␣
↪abs550aer, od550aer, rsdscs, hur' --input-path /lcrc/group/e3sm/e3sm_to_cmip/input/atm-
↪unified-eam-ncclimo --user-metadata /home/ac.tvo/E3SM-Project/CMIP6-Metadata/template.
↪json --tables-path /home/ac.tvo/PCMDI/cmip6-cmor-tables/Tables/ --serial
```

Python Execution

```
from e3sm_to_cmip.__main__ import main

args = [
    "--var-list",
    'pfull, phalf, tas, ts, psl, ps, sfcWind, huss, pr, prc, prsn, evspsbl, tauu, tauv,␣
↪hfls, clt, rlds, rlus, rsds, rsus, hfss, cl, clw, cli, clivi, clwvi, prw, rldscs, rlut,
↪ rlutcs, rsdt, rsuscs, rsut, rsutcs, rtmt, abs550aer, od550aer, rsdscs, hur',
    "--input",
    "/lcrc/group/e3sm/e3sm_to_cmip/input/atm-unified-eam-ncclimo",
    "--output",
    "../qa/tmp",
    "--tables-path",
    "/lcrc/group/e3sm/e3sm_to_cmip/cmip6-cmor-tables/Tables/",
    "--user-metadata",
    "/lcrc/group/e3sm/e3sm_to_cmip/template.json",
```

(continues on next page)

(continued from previous page)

```
    "--serial"
]

# `main()` creates an `E3SMtoCMIP` object and passes `args` to it, which sets the object␣
↪parameters to execute a run.
main(args)
```

Example 2 (info mode)

CLI Execution

```
e3sm_to_cmip --info -v prw, pr --input /p/user_pub/work/E3SM/1_0/historical/1deg_atm_60-
↪30km_ocean/atmos/native/model-output/day/ens1/v1/ --tables /home/vo13/PCMDI/cmip6-cmor-
↪tables/Tables/
```

Python Execution

```
from e3sm_to_cmip.__main__ import main

args = [
    "--info",
    "-v",
    "prw, pr",
    "--input",
    "/p/user_pub/work/E3SM/1_0/historical/1deg_atm_60-30km_ocean/atmos/native/model-
↪output/day/ens1/v1/",
    "--output",
    "../qa/tmp",
    "--tables-path",
    "/home/vo13/PCMDI/cmip6-cmor-tables/Tables/",
]

# `main()` creates an `E3SMtoCMIP` object and passes `args` to it, which sets object␣
↪parameters to execute a run.
main(args)
```

Example 3 (E3SMtoCMIP class inspection)

This process is useful for checking how `e3sm_to_cmip` interprets the CLI arguments, and which handlers are derived based on `--var-list`.

```
from e3sm_to_cmip.__main__ import E3SMtoCMIP

args = [
    "--var-list",
    'pfull, phalf, tas, ts, psl, ps, sfcWind, huss, pr, prc, prsn, evspsbl, tauu, tauv,␣
↪hfls, clt, rlds, rlus, rsds, rsus, hfss, cl, clw, cli, clivi, clwvi, prw, rldscs, rlut,
↪ rlutcs, rsdt, rsuscs, rsut, rsutcs, rtmt, abs550aer, od550aer, rsdscs, hur',
    "--input",
    "/lcrc/group/e3sm/e3sm_to_cmip/input/atm-unified-eam-ncclimo",
```

(continues on next page)

(continued from previous page)

```
    "--output",
    "../qa/tmp",
    "--tables-path",
    "/lcrc/group/e3sm/e3sm_to_cmip/cmip6-cmor-tables/Tables/",
    "--user-metadata",
    "/lcrc/group/e3sm/e3sm_to_cmip/template.json",
    "--serial"
]

run = E3SMtoCMIP(args)

# Now we can check the `E3SMtoCMIP` object attributes for the `run` variable.
print(run.handlers)
```