# CMOR

version 3.10

Charles Doutriaux
Karl E. Taylor
Chris Mauzey
Denis Nadeau
Paul J. Durack

*Last generated: June 03, 2025*

# Table of Contents

# Getting started overview

## Design Considerations and Overview

This document describes Version 3 of a software library called “Climate Model Output Rewriter” (CMOR3)[1] (page 8), written in C with access also provided via Fortran 90 and through Python[2] (page 8). CMOR is used to produce CF-compliant[3] (page 8) netCDF[4] (page 8) files. The structure of the files created by CMOR and the metadata they contain fulfill the requirements of many of the climate community’s standard model experiments (which are referred to here as “MIPs”[5] (page 8) and include, for example, AMIP, PMIP, APE, and IPCC [DN1] scenario runs).

CMOR was not designed to serve as an all-purpose writer of CF-compliant netCDF files, but simply to reduce the effort required to prepare and manage MIP model output. Although MIPs encourage systematic analysis of results across models, this is only easy to do if the model output is written in a common format with files structured similarly and with sufficient metadata uniformly stored according to a common standard. Individual modeling groups store their data in different ways, but if a group can read its own data, then it should easily be able to transform the data, using CMOR, into the common format required by the MIPs. The adoption of CMOR as a standard code for exchanging climate data will facilitate participation in MIPs because after learning how to satisfy the output requirements of one MIP, it will be easy to prepare output for other MIPs.

CMOR output has the following characteristics:

- Each file contains a single primary output variable (along with coordinate/grid variables, attributes and other metadata) from a single model and a single simulation (i.e., from a single ensemble member of a single climate experiment). This method of structuring model output best serves the needs of most researchers who are typically interested in only a few of the many variables in the MIP databases. Data requests can be satisfied by simply sending the appropriate file(s) without first extracting the individual field(s) of interest.
- There is flexibility in specifying how many time slices (samples) are stored in a single file. A single file can contain all the time-samples for a given variable and climate experiment, or the samples can be distributed in a sequence of files.
- Much of the metadata written to the output files is defined in MIP-specific tables of information, which in this document are referred to simply as “MIP tables”. These tables are JSON files that can be read by CMOR and are typically made available from MIP web sites. Because these tables contain much of the metadata that is useful in the MIP context, they are the key to reducing the programming burden imposed on the individual users contributing data to a MIP. Additional tables can be created as new MIPs are born.
- For metadata, different MIPs may have different requirements, but these are accommodated by CMOR, within the constraints of the CF convention and as specified in the MIP tables (e.g. CMIP6 MIP tables (https://github.com/PCMDI/cmip6-cmor-tables)).

- CMOR relies on NetCDF4 See unidata web page (http://www.unidata.ucar.edu/software/netcdf) to write the output files and can take advantage of its compression and chunking capabilities. CMOR3 outputs NetCDF4 files by default. For CMIP6, the NetCDF4/NC_CLASSIC_Model mode is used. As of CMOR 3.10, there is now support for zstandard compression and lossy compression via quantization (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#lossy-compression-via-quantization) . More on quantization in NetCDF4 here (https://docs.unidata.ucar.edu/netcdf-c/4.9.2/md__media_psf_Home_Desktop_netcdf_releases_v4_9_2_release_netcdf_c_docs_quantize.html) .
- CMOR also must be linked against the udunits2 library see http://www.unidata.ucar.edu/software/udunits/ (http://www.unidata.ucar.edu/software/udunits/), which enables CMOR to check that the units attribute is correct[6] (page 8). Finally CMOR3 must also be linked against the uuid library see http://www.ossp.org/pkg/lib/uuid (http://www.ossp.org/pkg/lib/uuid) in order to produce a unique tracking number for each file.

Although the CMOR output adheres to a fairly rigid structure, there is considerable flexibility allowed in the design of codes that write data through the CMOR functions. Depending on how the source data are stored, one might want to structure a code to read and rewrite the data through CMOR in several different ways. Consider, for example, a case where data are originally stored in “history” files that contain many different fields, but a single time sample. If one were to process several different fields through CMOR and one wanted to include many time samples per file, then it would usually be more efficient to read all the fields from the single input file at the same time, and then distribute them to the appropriate CMOR output files, rather than to process all the time-samples for a single field and then move on to the next field. If, however, the original data were stored already by field (i.e., one variable per file), then it would make more sense to simply loop through the fields, one at a time. The user is free to structure the conversion program in either of these ways (among others).

The following input files are typically needed by CMOR:

- The “User Input File” (e.g., CMIP6_input_example.json), which provides user-supplied metadata and configuration directives.
- A “controlled vocabulary file” (e.g., “CMIP6_CV.json), which concatenates into a single file most of the CMIP6 controlled vocabularies archived at https://github.com/WCRP-CMIP/CMIP6_CVs . This file is updated frequently as additional models and institutions register to participate in CMIP6.
- A “CMOR Table” (e.g., CMIP6_Amon.json), which provides for each variable that might be written by CMOR much of the required metatdata. It also provides additional information that CMOR uses to correctly write the data and to enable certain QC checks.
- A “Vertical Coordinate Formula Terms Table” (e.g., CMIP6_formula_terms.json)
- A “Coordinates Table” (e.g., CMIP6_coordinate.json) CMIP6
- CMIP6_grids.json supplements the Coordinates Table with axis information that is sometimes needed in the treatment of unstructured grids.

The files used by CMOR for CMIP6 are archived in https://github.com/PCMDI/cmip6-cmor-tables/tree/master/Tables (https://github.com/PCMDI/cmip6-cmor-tables/tree/master/Tables), and all but the CMIP6_input_example.json file must not be modifed by the user. The CMIP6_input_example.json file must be edited to accurately reflect the output being written by the user (but do not modify the lines of text appearing after the comment line, "#note_CV": " ** The following will be obtained from the CV and do not need to be defined here"). Note that the CMIP6_CV.json file found in https://github.com/PCMDI/cmip6-cmor-tables/tree/master/Tables (https://github.com/PCMDI/cmip6-cmor-tables/tree/master/Tables) is updated whenever new models and institutions are registered to participate in CMIP6.

Converting data with CMOR typically involves the following steps (with the CMOR function names given in parentheses):

- Initialize CMOR and specify where output will be written and how error messages will be handled (cmor_setup).
- Provide information directing where output should be placed and identifying the data source, project name, experiment, etc. (cmor_dataset_json). User need to provide a User Input CMOR file to define each attribute.
- Set any additional "dataset" (i.e. global) attributes (cmor_set_cur_dataset function). Note that all CMIP6 attributes can also be defined in the CMOR input user JSON file (cmor_dataset_json).
- Define the axes (i.e., the coordinate values) associated with each of the dimensions of the data to be written and obtain "handles", to be used in the next step, which uniquely identify the axes (cmor_axis).
- In the case of non-Cartesian longitude-latitude grids or for "station data", define the grid and its mapping parameters (cmor_grid and cmor_set_grid_mapping)
- Define the variables to be written by CMOR, indicate which axes are associated with each variable, and obtain "handles", to be used in the next step, which uniquely identify each variable (cmor_variable). For each variable defined, this function fills internal table entries containing file attributes passed by the user or obtained from a MIP table, along with coordinate variables and other related information. Thus, nearly all of the file's metadata is collected during this step.
- Write an array of data that includes one or more time samples for a defined variable (cmor_write). This step will typically be repeated to output additional variables or to append additional time samples of data.
- Close one or all files created by CMOR (cmor_close)

There is an additional function (cmor_zfactor), which enables one to define metadata associated with dimensionless vertical coordinates.

CMOR was designed to reduce the effort required of those contributing data to various MIPs. An important aim was to minimize any transformations that the user would have to perform on their original data structures to meet the MIP requirements. Toward this end, the code allows the following flexibility (with the MIP requirements obtained by CMOR from the appropriate MIP table and automatically applied):

- The input data can be structured with dimensions in any order and with coordinate values either increasing or decreasing monotonically; CMOR will rearrange them to meet the MIP's requirements before writing out the data.
- The input data and coordinate values can be provided in an array declared to be whatever "type" is convenient for the user (e.g., in the case of coordinate data, the user might pass type "real" values (32-bit floating-point numbers on most platforms) even though the output will be written type double (64-bit IEEE floating-point); CMOR will transform the data to the required type before writing.
- The input data can be provided in units different from what is required by a MIP. If those units can be transformed to the correct units using the udunits (version 2) software (see udunits)[http://www.unidata.ucar.edu/software/udunits/], then CMOR performs the transformation before writing the data. Otherwise, CMOR will return an error. Time units are handled via the built-in cdtime interface [7] (page 9).
- So-called "scalar dimensions" (sometimes referred to as "singleton dimensions") are automatically inserted by CMOR. Thus, for example, the user can provide surface air temperature (at 2 meters) as a function of longitude, latitude, and time, and CMOR adds as a "coordinate" attribute the "height" dimension, consistent with the metadata requirements of CF. If the model output does not conform to the MIP requirements (e.g., carries temperature at 1.5 m instead of 2 m), then the user can override the MIP table specifications.

The code does not, however, include a capability to interpolate data, either in the vertical or horizontally. If data originally stored on model levels, is supposed to be stored on standard pressure levels, according to MIP specifications, then the user must interpolate before passing the data to CMOR.

The output resulting from CMOR is "self-describing" and includes metadata summarized below, organized by attribute type (global, coordinate, or variable attributes) and by its source (specified by the user or in a MIP table, or generated by CMOR).

*Global attributes typically provided by the MIP table or generated by CMOR:*

- *title*, identification of the project, experiment, and table.
- *Conventions*, ('CF-1.4')
- *history*, any user-provided history along with a "timestamp" generated by CMOR and a statement that the data conform to both the CF standards and those of a particular MIP.
- *activity_id*, scientific project that inspired this simulation (e.g., CMIP6)
- *table_id*, MIP table used to define variable.
- *data_specs_version* Base on the latest CMIP6-Datarequest latest database version.
- *mip_era*, define what cycle of CMIP dictates the experiment and data specificiation.
- *experiment*, a long name title for the experiment.
- *realm(s)* to which the variable belongs (e.g., ocean, land, atmosphere, etc.).
- *tracking_id*, a unique identification string generated by uuid, which is useful at least within the ESG distributed data archive.

- *cmor_version*, version of the library used to generate the files.
- *frequency*, the approximate time-sampling interval for a time-series of data.
- *creation_date*, the date and time (UTZ) that the file was created.
- *product*, a descriptive string that distinguishes among various model data products.

*Global attributes typically provided by the user in a call to a CMOR function:*

- *institution*, identifying the modeling center contributing the output.
- *institute_id*, a shorter identifying name of the modeling center (which would be appropriate for labeling plots in which results from many models might appear).
- *source*, identifying the model version that generated the output.
- *contact*, providing the name and email of someone responsible for the data
- *source_id*, an acronym that identifies the model used to generate the output.
- *experiment_id*, a short name for the experiment.
- *history*, providing an “audit trail” for the data, which will be supplemented with CMOR-generated information described above.
- *references*, typically containing documentation of the model and the model simulation.
- *comment*, typically including initialization and spin-up information for the simulation.
- *realization_index*, an integer distinguishing among simulations that differ only from different equally reasonable initial conditions. This number should be greater than or equal to 1.
- *initialization_index*, an integer distinguishing among simulations that differ only in the method of initialization. This number should be greater than or equal to 1.
- *physics_index*, an integer indicating which of several closely related physics versions of a model produced the simulation.
- *parent_experiment_id*, a string indicating which experiment this branches from. For CMIP6 this should match the short name of the parent experiment id.
- *parent_experiment_rip*, a string indicating which member of an ensemble of parent experiment runs this simulation branched from.
- *branch_time*, time in parent experiment when this simulation started (in the units of the parent experiment).

Note: additional global attributes can be added by the user via the cmor_set_cur_dataset_attribute function (see below).

*Coordinate attributes typically provided by a MIP table or generated by CMOR:*

- *standard_name*, as defined in the CF standard name table.
- *units*, specifying the units for the coordinate variable.
- *axis*, indicating whether axis is of type x, y, z, t, or none of these.
- *bounds*, (when appropriate) indicating where the cell bounds are stored.

- *positive*, (when appropriate) indicating whether a vertical coordinate increases upward or downward.
- *formula_terms*, (when appropriate) providing information needed to transform from a dimensionless vertical coordinate to the actual location (e.g., from sigma-level to pressure).
- Coordinate or grid mapping attributes typically provided by the user in a call to a CMOR function:*
- *calendar*, (when appropriate) indicating the calendar type assumed by the model.
- *grid_mapping_name* and the names of various mapping parameters, when necessary to describe grids other than lat-lon. See CF conventions at: http://cfconventions.org/Data/cf-conventions/cf-conventions-1.7/cf-conventions.html#appendix-grid-mappings (http://cfconventions.org/Data/cf-conventions/cf-conventions-1.7/cf-conventions.html#appendix-grid-mappings)
- Variable attributes typically provided by a MIP table or generated by CMOR:*
- *standard_name* as defined in the CF standard name table.
- *units*, specifying the units for the variable.
- *long_name*, describing the variable and useful as a title on plots.
- *missing_value* and *_FillValue*, specifying how missing data will be identified.
- *cell_methods*, (when appropriate) typically providing information concerning calculation of means or climatologies, which may be supplemented by information provided by the user.
- *cell_measures*, when appropriate, indicates the names of the variables containing cell areas and volumes.
- *comment*, providing clarifying information concerning the variable (e.g., whether precipitation includes both liquid and solid forms of precipitation).
- *history*, indicating what CMOR has done to the user supplied data (e.g., transforming its units or rearranging its order to be consistent with the MIP requirements)
- *coordinates*, (when appropriate) supplying either scalar (singleton) dimension information or the name of the labels containing names of geographical regions.
- *flag_values* and *flag_meanings*
- *modeling_realm*, providing the realm associated to the variable (ocean, land, aerosol, SeaIce, LandIce, …)

*Variable attributes typically provided by the user in a call to a CMOR function:*

- *grid_mappingi*
- *original_name*, containing the name of the variable as it is known at the user's home institution.i*
- *original_units*, the units of the data passed to CMOR.
- *history*, (when appropriate) information concerning processing of the variable prior to sending it to CMOR. (This information may be supplemented by further history information generated by

CMOR.)

- *comment*, (when appropriate) providing miscellaneous information concerning the variable, which will supplement any comment contained in the MIP table.

As is evident from the above summary of metadata, a substantial fraction of the information is defined in the MIP tables, which explains why writing MIP output through CMOR is much easier than writing data without the help of the MIP tables. Besides the attribute information, the MIP tables also include information that controls the structure of the output and allows CMOR to apply some rudimentary quality assurance checks. Among this ancillary information in the MIP tables is the following:

- The direction each coordinate should be stored when it is output (i.e., either in order of increasing or decreasing values). The user need not be concerned with this since, if necessary, CMOR will reorder the coordinate values and the data.
- The acceptable values for coordinates (e.g., for a pressure coordinate axis, for example, perhaps the WCRP standard pressure levels).
- The acceptable values for various arguments passed to CMOR functions (e.g., acceptable calendars, experiment i.d.'s, etc.)
- The "type" of each output array (whether real, double precision, or integer). The user need not be concerned with this since, if necessary, CMOR will convert the data to the specified type.
- The order of the dimensions for output arrays. The user need not be concerned with this since, if necessary, CMOR will reorder the data consistent with the specified dimension order.
- The normally applied values for "scalar dimensions" (i.e., "singleton dimensions").
- The range of acceptable values for output arrays.
- The acceptable range for the spatial mean of the absolute value of all elements in output arrays.
- The minimal global attributes required.

[1] CMOR is pronounced "C-more", which suggests that CMOR should enable a wide community of scientists to "see more" climate data produced by modeling centers around the world. CMOR also reminds us of Ecinae Corianus, the revered ancient Greek scholar, known to his friends as "Seymour". Seymour spent much of his life translating into Greek nearly all the existing climate data, which had originally been recorded on largely inscrutable hieroglyphic and cuneiform tablets. His resulting volumes, organized in a uniform fashion and in a language readable by the common scientists of the day, provided the basis for much subsequent scholarly research. Ecinae Corianus was later indirectly honored by early inhabitants of the British Isles who reversed the spelling of his name and used the resulting string of letters, grouped differently, to form new words referring to the major elements of climate.

[2] CMOR1 was written in Fortran 90 with access also provided through Python.

[3] See http://www.cgd.ucar.edu/cms/eaton/cf-metadata

[4] See http://my.unidata.ucar.edu/content/software/netcdf/

[5] "MIP" is an acronym for "model intercomparison project".

[6] CMOR1 was linked to an earlier version of the netCDF library and udunits was optional.

[7] Cdtime is now built into CMOR. Therefore linking against cdms is no longer necessary.

## Preliminary notes

In the following, all arguments should be passed using keywords (to improve readability and flexibility in ordering the arguments). Those arguments appearing below that are followed by an equal sign may be optional and, if not passed by the user, are assigned the default value that follows the equal sign. The information in a MIP-specific input table determines whether or not an argument shown in brackets is optional or required, and the table provides MIP-specific default values for some parameters. All arguments not in brackets and not followed by an equal sign are always required.

Three versions of each function are shown below. The first one is for Fortran (green text) the second for C (blue text), and the third for Python (orange text). In the following, text that applies to only one of the coding languages appears in the appropriate color.

Some of the arguments passed to CMOR (e.g., names of variables and axes are only unambiguously defined in the context of a specific CMOR table, and in the Fortran version of the functions this is specified by one of the function arguments, whereas in the C and Python versions it is specified through a call to cmor_load_table and cmor_set_table.

All functions are type “integer”. If a function results in an error, an “exception” will be raised in the Python version (otherwise None will be returned), and in either the Fortran or C versions, the error will be indicated by the integer returned by the function itself. In C an integer other than 0 will be returned, and in Fortran errors will result in a negative integer (except in the case of cmor_grid, which will return a positive integer).

If no error is encountered, some functions will return information needed by the user in subsequent calls to CMOR. In almost all cases this information is indicated by the value of a single integer that in Fortran and Python is returned as the value of the function itself, whereas in C it is returned as an output argument). There are two cases in the Fortran version of CMOR, however, when a string argument may be set by CMOR (cmor_close and cmor_create_output_path). These are the only cases when the value of any of the Fortran function’s arguments might be modified by CMOR.

# CMOR Application program interface (API)

cmor_setup()

Fortran: error_flag = cmor_setup(inpath='./', netcdf_file_action=CMOR_PRESERVE, set_verbosity=CMOR_NORMAL, exit_control=CMOR_NORMAL, logfile, create_subdirectories)

C: error_flag = cmor_setup(char *inpath, int *netcdf_file_action, int *set_verbosity, int *exit_control, char *logfile, int *create_subdirectories)

Python: setup(inpath='.', netcdf_file_action=CMOR_PRESERVE, set_verbosity=CMOR_NORMAL, exit_control=CMOR_NORMAL, logfile=None, create_subdirectories=1)

*Description:* Initialize CMOR, specify path to MIP table(s) that will be read by CMOR, specify whether existing output files will be overwritten, and specify how error messages will be handled

*Arguments:*

- **[inpath]** = a character string specifying the path to the directory where the needed MIP-specific tables reside.
- **[netcdf_file_action]** = controls handling of existing netCDF files. If the value passed is CMOR_REPLACE, a new file will be created; any existing file with the same name as the one CMOR is trying to create will be overwritten. If the value is CMOR_APPEND, an existing file will be appended; if the file does not exist, it will be created. If the value is CMOR_PRESERVE, a new file will be created unless a file by the same name already exists, in which case the program will error exit.[8] To generate a NetCDF file in the "CLASSIC" NetCDF3 format, a "_3" should be appended to the above parameters (e.g., CMOR_APPEND would become CMOR_APPEND_3). To generate a NetCDF file in the "CLASSIC" NetCDF4 format, a "_4" should be appended to the above parameters (e.g., CMOR_APPEND would become CMOR_APPEND_4), this allows the user to take advantage of NetCDF4 compression and chunking capabilities. The default values (no underscore) are aliased to the _4 values (satisfying the requirements of CMIP6).
- **[set_verbosity]** controls how informational messages and error messages generated by CMOR are handled. If set_verbosity=CMOR_NORMAL, errors and warnings will be sent to the standard error device (typically the user's screen). If verbosity=CMOR_QUIET, then only error messages will be sent (and warnings will be suppressed).
- **[exit_control]** determines if errors will trigger program to exit:
- CMOR_EXIT_ON_MAJOR = stop only on critical error;
- CMOR_NORMAL = stop only if severe errors;
- CMOR_EXIT_ON_WARNING = stop even after minor errors detected.
- **[logfile]** where CMOR will write its messages – default is "standard error" (stderr).
- **[create_subdirectories]** do we want to create the correct path subdirectory structure or

simply dump the files wherever cmor_dataset will point to.

Returns upon success:

- Fortran: 0
- C: 0
- Python: None

---

cmor_dataset_json()

Fortran: cmor_dataset_json(filename)

C: cmor_dataset_json(char *name)

Python: dataset_json(name)

*Description*: This function provides information to CMOR that is common to all output files that will be written. The “dataset” defined by this function refers to some or all of the output from a single model simulation (i.e., output from a single realization of a single experiment from a single model). Only one dataset can be defined at any time, but the dataset can be closed (by calling cmor_close()), and then another dataset can be defined by calling cmor_dataset. Note that after a new dataset is defined, all axes and variables must be defined; axes and variables defined earlier are not associated with the new dataset.

*Arguments*:

- **name**: JSON file which contains all information needed by CMOR in the form of key:value. Here is an example: CMIP6_input_example.json (https://github.com/PCMDI/cmip6-cmor-tables/blob/main/Tables/CMIP6_input_example.json)

Returns upon success:

- Fortran: 0
- C: 0
- Python: 0

---

cmor_set_cur_dataset_attribute()

Fortran: error_flag = cmor_set_cur_dataset_attribute(name,value)

C: error_flag = cmor_set_cur_dataset_attribute(char *name, char *value, int optional)

Python: set_cur_dataset_attribute(name,value)

*Description*: Associate a global attribute with the current dataset. In CMIP5, this function can be called to set, for example, “institute_id”, “initialization” and “physics”.

*Arguments*:

- **name** = name of the global attribute to set.
- **value** = character string containing the value of this attribute.

- **optional** = an argument that is ignored. (Internally, CMOR calls this function and needs this argument.)

Returns upon success:

- Fortran: 0
- C: 0
- Python: None

---

cmor_get_cur_dataset_attribute()

Fortran: error_flag = cmor_get_cur_dataset_attribute(name,result)

C: error_flag = cmor_get_cur_dataset_attribute(char *name, char *result)

Python: result = get_cur_dataset_attribute(name)

*Description*: Retrieves a global attribute associated with the current dataset.

*Arguments*:

- **name** = name of the global attribute to retrieve.
- **result** = string (or pointer to a string), which is returned by the function and contains the retrieved global attribute (not for Python).

Returns upon success:

- Fortran: 0
- C: 0
- Python: None

---

cmor_has_cur_dataset_attribute()

Fortran: error_flag = cmor_has_cur_dataset_attribute(name)

C: error_flag = cmor_has_cur_dataset_attribute(char *name)

Python: error_flag = has_cur_dataset_attribute(name)

*Description:* Determines whether a global attribute is associated with the current dataset.

*Arguments:*

- **name** = name of the global attribute of interest.

Returns:

- a negative integer if an error is encountered; otherwise returns 0.
- 0 upon success
- True if the attribute exists, False otherwise.

---

cmor_load_table()

Fortran: table_id = cmor_load_table(table)

C: error_flag = cmor_load_table(char *table, int *table_id)

Python: table_id = load_table(table)

*Description:* Loads a table and returns a “handle” (table_id) to use later when defining CMOR components. CMOR will look for the table first following the path as specified by the “table” argument passed to this function. If it doesn’t find a file there it will prepend the outpath defined in calling cmor_dataset. If it still doesn’t find it, it will use the “prefix” where the library CMOR is to be installed (from configure time) followed by share (e.g /usr/local/cmor/share). If it stills fails an error will be raised.

cmor_set_table()

Fortran: cmor_set_table(table_id)

C: error_flag = cmor_set_table(int table_id)

Python: table_id = set_table(table_id)

*Description:* Sets the table referred to by table_id as the table to obtain needed information when defining CMOR components (variables, axes, grids, etc…).

cmor_axis()

Fortran: axis_id = cmor_axis([table], table_entry, units, [length], [coord_vals], [cell_bounds], [interval])

C: error_flag = cmor_axis(int *axis_id, char *table_entry, char *units, int length, void *coord_vals, char type, void *cell_bounds, int cell_bounds_ndim, char *interval)

Python: axis_id = axis(table_entry, length=??, coord_vals=None, units=None, cell_bounds=None, interval=None)

*Description:* Define an axis and pass the coordinate values associated with one of the dimensions of the data to be written. This function returns a “handle” (axis_id) that uniquely identifies the axis to be written. The axis_id will subsequently be passed by the user to other CMOR functions. The cmor_axis function will typically be repeatedly invoked to define all axes. The axis specified by the table_entry argument must be found in the currently “set” CMOR table, as specified by the cmor_load_table and cmor_set_table functions, or as an option, it can be provided in the Fortran version (for backward compatibility) by the now deprecated “table” keyword argument. There normally is no need to call this function in the case of a singleton (scalar) dimension unless the MIP recommended (or required) coordinate value (or cell_bounds) are inconsistent with what the user can supply, or unless the user wants to define the “interval” attribute. When defining a time axis in CMOR “Append mode” (in case the file already existed before a call to cmor_setup), time values and bounds should be set to NULL and instead be passed via cmor_write when writing data.

*Arguments:*

- **[table]** = character string containing the filename of the MIP-specific table where the axis defined here appears. (e.g., ‘CMIP5_table_Amon’, ‘IPCC_table_A1’, ‘AMIP_table_1a’, ‘AMIP_table_2’, ‘CMIP_table_2’, etc.). In CMOR2 this is an optional argument and is deprecated because the table can be specified through the cmor_load_table and cmor_set_table functions.
- **axis_id** = the “handle”: a positive integer returned by CMOR, which uniquely identifies the axis stored in this call to cmor_axis and subsequently can be used in calls to cmor_write.
- **table_entry** = name of the axis (as it appears in the MIP table) that will be defined by this function. units = units associated with the coordinates passed in coord_vals and cell_bounds. (These are the units of the user’s coordinate values, which, if CMOR is built with udunits (as is required in version 2), may differ from the units of the coordinates written to the netCDF file by CMOR.) These units must be recognized by udunits or must be identical to the units specified in the MIP table. In the case of a dimensionless vertical coordinate or in the case of a non-numerical axis (like geographical region), either set units=’none’, or, optionally, set units=’1’.
- **[length]** = integer specifying the number of elements that CMOR should extract from the coord_vals array (normally length will be the size of the array itself). For a simple “index axis” (i.e., an axis without coordinate values), this specifies the length of the dimension. In the Fortran and Python versions of the function, this argument is not always required (except in the case of a simple index axis); if omitted “length” will be the size of the coord_vals array,
- **[coord_vals]** = 1-d array (single precision float, double precision float, or, for labels, character strings) containing coordinate values, ordered consistently with the data array that will be passed by the user to CMOR through function cmor_write (see documentation below). This argument is required except if: 1) the axis is a simple “index axis” (i.e., an axis without coordinate values), or 2) for a time coordinate, the user intends to pass the coordinate values when the cmor_write function is called. Note that the coordinate values must be ordered monotonically, so, for example, in the case of longitudes that might have the values, 0., 10., 20, … 170., 180., 190., 200., … 340., 350., passing the (equivalent) values, 0., 10., 20, … 170., 180., -170., -160., … -20., -10. is forbidden. In the case of time-coordinate values, if cell bounds are also passed, then CMOR will first check that each coordinate value is not outside its associated cell bounds; subsequently, however, the user-defined coordinate value will be replaced by the mid-point of the interval defined by its bounds, and it is this value that will be written to the netCDF file. In the case of character string coord_vals there are no cell_bounds, but for the C version of the function, the argument cell_bounds_ndim is used to specify the length of the strings in the coord_val array (i.e., the array will be dimensioned [length][cell_bounds_ndim]).
- **type** = type of the coord_vals/bnds passed, which can be ‘d’ (double), ‘f’ (float), ‘l’ (long) or ‘i’ (int).
- **[cell_bounds]** = 1-d or 2-d array (of the same type as coord_vals) containing cell bounds, which should be in the same units as coord_vals (specified in the “units” argument above) and should be ordered in the same way as coord_vals. In the case of a 1-d array, the size is one more than the size of coord_vals and the cells must be contiguous. In the case of a 2-d array, it is dimensioned (2, n) where n is the size of coord_vals (see CF standard document, http://www.cgd.ucar.edu/cms/eaton/cf-metadata, for further information). This argument may be omitted when cell bounds are not required. It must be omitted if coord_vals is omitted.

- **cell_bounds_ndim** = This argument only appears in the the C version of this function. Except in the case of a character string axis, it specifies the rank of the cell_bounds array: if 1, the bounds array will contain n+1 elements, where n is length of coords and the cells must be contiguous, whereas if 2, the dimension will be (n,2) in C order. Pass 0 if no cell_bounds values have been passed. In the special case of a character string axis, this argument is used to specify the length of the strings in the coord_val array (i.e., the array will be dimensioned [length] [cell_bounds_ndim]).
- **[interval]** = Supplemental information that will be included in the cell_methods attribute, which is typically defined for the time axis in order to describe the sampling interval. This string should be of the form: “value unit comment: anything” (where “comment:” and anything may always be omitted). For monthly mean data sampled every 15 minutes, for example, interval = “15 minutes”.

Returns:

- Fortran: a negative integer if an error is encountered; otherwise returns a positive integer (the “handle”) uniquely identifying the axis ..
- C: 0 upon success.
- Python: upon success, a positive integer (the “handle”) uniquely identifying the axis, or if an error is encountered an exception is raised.

---

cmor_grid()

Fortran: grid_id = cmor_grid(axis_ids, latitude, longitude, [latitude_vertices], [longitude_vertices], [area])

C: error_flag = cmor_grid(int *grid_id, int ndims, int *axis_ids, char type, void *latitude, void *longitude, int nvertices, void *latitude_vertices, void *longitude_vertices, void *area)

Python: grid_id = grid( axis_ids, latitude, longitude, latitude_vertices=None, longitude_vertices=None, area=None)

*Description:* Define a grid to be associated with data, including the latitude and longitude arrays. The grid can be structured with up to 6 dimensions. These dimensions, which may be simple “index” axes, must be defined via cmor_axis prior to calling cmor_grid. This function returns a “handle” (grid_id) that uniquely identifies the grid (and its data/metadata) to be written. The grid_id will subsequently be passed by the user to other CMOR functions. The cmor_grid function will typically be invoked to define each grid necessary for the experiment (e.g ocean grid, vegetation grid, atmosphere grid, etc…). There is no need to call this function in the case of a Cartesian lat/lon grid. In this case, simply define the latitude and longitude axes and pass their id’s (“handles”) to cmor_variable.

*Arguments:*

- **grid_id** = the “handle”: a positive integer returned by CMOR, which uniquely identifies the grid defined in this call to CMOR and subsequently can be used in calls to CMOR.
- **ndims** = number of dimensions needed to define the grid. Namely the number of elements from axis_ids that will be used.
- **axis_ids** = array containing the axis_s returned by cmor_axis when defining the axes constituing the grid.

- **latitude** = array containing the grid's latitude information (ndim dimensions)
- **longitude** = array containing the grid's longitude information (ndim dimensions)
- **[latitude_vertices]** = array containing the grid's latitude vertices information (ndim+1 dimensions). The vertices dimension must be the fastest varying dimension of the array (i.e first one in Fortran, last one in C, last one in Python)
- **[longitude_vertices]** = array containing the grid's longitude vertices information (ndim+1 dimensions). The vertices dimension must be the fastest varying dimension of the array (i.e first one in Fortran, last one in C, last one in Python)
- **[area]** = array containing the grid's area information (ndim)

Returns:

- Fortran: a positive integer if an error is encountered; otherwise returns a negative integer (the "handle") uniquely identifying the grid.
- C: 0 upon success.
- Python: upon success, a positive integer (the "handle") uniquely identifying the axis, or if an error is encountered an exception is raised.

---

cmor_set_grid_mapping()

Fortran: error_flag = cmor_set_grid_mapping(grid_id, mapping_name, parameter_names, parameter_values, parameter_units)

C: error_flag = cmor_set_grid_mapping(int grid_id, char *mapping_name, int nparameters, char **parameter_names, int lparameters, double parameter_values[], char **parameter_units, int lunits)

Python: set_grid_mapping(grid_id, mapping_name, parameter_names, parameter_values=None, parameter_units=None)

*Description*: Define the grid mapping parameters associated with a grid (see CF conventions for more info on which parameters to set). Check validity of parameter names and units. Additional mapping names and parameter names can be defined via the MIP table.

*Arguments*:

- **grid_id** = the "handle" returned by a previous call to cmor_grid, indicating which grid the mapping parameters should be associated with.
- **mapping_name** = name of the mapping (see CF conventions (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#appendix-grid-mappings) for a list of grid mapping names). This name dictates which parameters should be set and for some parameters restricts their possible values or range. New mapping names can be added via MIP tables. Below is the list of valid grid mapping names.
    - albers_conical_equal_area (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_albers_equal_area)

- azimuthal_equidistant (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#azimuthal-equidistant)
- geostationary (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_geostationary_projection)
- lambert_azimuthal_equal_area (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#lambert-azimuthal-equal-area)
- lambert_conformal_conic (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_lambert_conformal)
- lambert_cylindrical_equal_area (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_lambert_cylindrical_equal_area)
- latitude_longitude (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_latitude_longitude)
- mercator (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_mercator)
- oblique_mercator (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_oblique_mercator)
- orthographic (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_orthographic)
- polar_stereographic (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#polar-stereographic)
- rotated_latitude_longitude (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_rotated_pole)
- sinusoidal (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_sinusoidal)
- stereographic (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_stereographic)
- transverse_mercator

(https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_transverse_mercator)

- vertical_perspective (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#vertical-perspective)

- **nparameters** = number of parameters set.
- **parameter_names** = array (list for Python) of strings containing the names of the parameters to set. In the case of "standard_parallel", CF allows either 1 or 2 parallels to be specified (i.e. the attribute standard_parallel may be an array of length 2). In the case of 2 parallels, CMOR requires the user to specify these as separate parameters, named standard_parallel_1 and standard_parallel_2, but then the two parameters will be stored in an array, consistent with CF. In the case of a single parallel, the name standard_parallel should be specified. In the C version of this function, parameter_names is declared of length [nparameters][lparameters], where lparameters in the length of each string array element (see below). In Python parameter_names can be defined as a dictionary containing the keys that represent the parameter_names. The value associated with each key can be either a list [float, str] (or [str, float]) representing the value/units of each parameter, or another dictionary containing the keys "value" and "units". If these conditions are fulfilled, then parameter_units and parameter_values are optional and would be ignored if passed.
- **lparameters** = length of each element of the string array. If, for example, parameter_names includes 5 parameters, each 24 characters long (i.e., it is declared [5][24]), you would pass lparameters=24.
- **parameter_values** = array containing the values associated with each parameter. In Python this is optional if parameter_names is a dictionary containing the values and units.
- **parameter_units** = array (list for Python) of string containing the units of the parameters to set. In C parameter_units is declared of length [nparameters][lunits]. In Python it is optional if parameter_names is a dictionary containing the value and units.
- **lunits** = length of each elements of the units string array (e.g., if parameters_units is declared [5][24], you would pass 24 because each elements has 24 characters).

Returns upon success:

- Fortran: 0
- C: 0
- Python: None

---

### cmor_set_crs()

Fortran: error_flag = cmor_set_crs(grid_id, mapping_name, parameter_names, parameter_values, parameter_units, text_parameter_names, text_parameter_values)

C: error_flag = cmor_set_crs(int grid_id, char *mapping_name, int nparameters, char **parameter_names, int lparameters, double parameter_values[], char **parameter_units, int lunits, int ntextparameters, char **text_parameter_names, int ltextparameters, char *text_parameter_values, int text_parameter_length[])

Python: set_crs(grid_id, mapping_name, parameter_names, parameter_values=None, parameter_units=None)

*Description*: Define the Coordinate Reference System (CRS) variable that contains grid mapping parameters associated with a grid (see CF conventions for more info on which parameters to set). Similar to cmor_set_grid_mapping (page 16), but allows for the setting of text-based parameters.

*Note on* `crs_wkt` : This function allows for the setting of a `crs_wkt` (https://cfconventions.org/cf-conventions/cf-conventions.html#use-of-the-crs-well-known-text-format) text attribute, meant to hold a Well-known Text (WKT) formatted string. However, CMOR will not validate the contents of this string.

*Arguments*:

- **grid_id** = the “handle” returned by a previous call to cmor_grid, indicating which grid the mapping parameters should be associated with.
- **mapping_name** = name of the mapping (see CF conventions (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#appendix-grid-mappings) for a list of grid mapping names). This name dictates which parameters should be set and for some parameters restricts their possible values or range. New mapping names can be added via MIP tables. Below is the list of valid grid mapping names.
    - albers_conical_equal_area (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_albers_equal_area)
    - azimuthal_equidistant (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#azimuthal-equidistant)
    - geostationary (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_geostationary_projection)
    - lambert_azimuthal_equal_area (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#lambert-azimuthal-equal-area)
    - lambert_conformal_conic (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_lambert_conformal)
    - lambert_cylindrical_equal_area (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_lambert_cylindrical_equal_area)
    - latitude_longitude

(https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_latitude_longitude)

- mercator (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_mercator)
- oblique_mercator (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_oblique_mercator)
- orthographic (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_orthographic)
- polar_stereographic (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#polar-stereographic)
- rotated_latitude_longitude (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_rotated_pole)
- sinusoidal (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_sinusoidal)
- stereographic (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_stereographic)
- transverse_mercator (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#_transverse_mercator)
- vertical_perspective (https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#vertical-perspective)

- **nparameters** = number of parameters set.
- **parameter_names** = array (list for Python) of strings containing the names of the parameters to set. In the case of “standard_parallel”, CF allows either 1 or 2 parallels to be specified (i.e. the attribute standard_parallel may be an array of length 2). In the case of 2 parallels, CMOR requires the user to specify these as separate parameters, named standard_parallel_1 and standard_parallel_2, but then the two parameters will be stored in an array, consistent with CF. In the case of a single parallel, the name standard_parallel should be specified. In the C version of this function, parameter_names is declared of length [nparameters][lparameters], where lparameters in the length of each string array element (see below). In Python, parameter_names can be defined as a dictionary containing the keys that represent the parameter_names. The value associated with each key can be either a list [float, str] (or [str, float]) representing the

value/units of each parameter, or another dictionary containing the keys “value” and “units”. If these conditions are fulfilled, then parameter_units and parameter_values are optional and would be ignored if passed.

- **lparameters** = length of each element of the parameter_names string array. If, for example, parameter_names includes 5 parameters, each 24 characters long (i.e., it is declared [5][24]), you would pass lparameters=24.
- **parameter_values** = array containing the values associated with each parameter. In Python, parameter_values is ignored if parameter_names is a dictionary containing the values and units.
- **parameter_units** = array (list for Python) of string containing the units of the parameters to set. In C parameter_units is declared of length [nparameters][lunits]. In Python, parameter_units is ignored if parameter_names is a dictionary containing the value and units.
- **lunits** = length of each elements of the units string array (e.g., if parameters_units is declared [5][24], you would pass 24 because each elements has 24 characters).
- **ntextparameters** = number of text parameters set.
- **text_parameter_names** = array (list for Python) of strings containing the names of the text parameters to set. In the C version of this function, text_parameter_names is declared of length [ntextparameters][ltextparameters], where ltextparameters is the length of each string array element (see below). In Python, parameter_names can be defined as a dictionary containing the keys that represent the text parameter names. The value associated with each key can be a string value of each text parameter.
- **ltextparameters** = length of each element of the text_parameter_names string array. If, for example, text_parameter_names includes 5 parameters, each 24 characters long (i.e., it is declared [5][24]), you would pass ltextparameters=24.
- **text_parameter_values** = array containing the string values associated with each text parameter. In the C version of this function, this array will contain null-terminated strings that are concatenated end-to-end in the same order as their names in text_parameter_names and their lengths in text_parameter_length. In Python, parameter_values is ignored if parameter_names is a dictionary containing the text values.
- **text_parameter_length** = array containing the length of each string value in text_parameter_values.

Returns upon success:

- Fortran: 0
- C: 0
- Python: None

---

cmor_time_varying_grid_coordinate()

Fortran: coord_var_id = cmor_time_varying_grid_coordinate(grid_id, table_entry, units, missing_value)

C: error_flag = cmor_time_varying_grid_coordinate(int *coord_var_id, int grid_id, char *table_entry, char *units, char type, void *missing, [int *coordinate_type])

Python: coord_var_id = time_varying_grid_coordinate(grid_id, table_entry, units, [missing_value])

*Description*: Define a grid to be associated with data, including the latitude and longitude arrays. Note that in CMIP5 this function must be called to store the variables called for in the cf3hr MIP table. The grid can be structured with up to 6 dimensions. These dimensions, which may be simple “index” axes, must be defined via cmor_axis prior to calling cmor_grid. This function returns a “handle” (grid_id) that uniquely identifies the grid (and its data/metadata) to be written. The grid_id will subsequently be passed by the user to other CMOR functions. The cmor_grid function will typically be invoked to define each grid necessary for the experiment (e.g., ocean grid, vegetation grid, atmosphere grid, etc.). There is no need to call this function in the case of a Cartesian lat/lon grid. In this case, simply define the latitude and longitude axes and pass their id’s (“handles”) to cmor_variable.

*Arguments:*

- **coord_var_id** = the “handle”: a positive integer returned by this function, which uniquely identifies the variable and can be used in subsequent calls to CMOR.
- **grid_id** = the value returned by cmor_grid when the grid was created.
- **table_entry** = name of the variable (as it appears in the MIP table) that this function defines.
- **units** = units of the data that will be passed to CMOR by function cmor_write. These units may differ from the units of the data output by CMOR. Whenever possible, this string should be interpretable by udunits (see http://my.unitdata.ucar.edu/content/software/udunits/). In the case of dimensionless quantities the units should be specified consistent with the CF conventions, so for example: percent, units=’%’; for a fraction, units=’1’; for parts per million, units=’1e-6’, etc.).
- **type** = type of the missing_value, which must be the same as the type of the array that will be passed to cmor_write. The options are: ‘d’ (double), ‘f’ (float), ‘l’ (long) or ‘i’ (int).
- **[missing_value]** = scalar that is used to indicate missing data for this variable. It must be the same type as the data that will be passed to cmor_write. This missing_value will in general be replaced by a standard missing_value specified in the MIP table. If there are no missing data, and the user chooses not to declare the missing value, then this argument may be omitted.
- **[coordinate_type]** = place holder for future implementation, unused, pass NULL

*Returns*:

- Fortran: a positive integer if an error is encountered; otherwise returns a negative integer (the “handle”) uniquely identifying the grid.
- C: 0 upon success.
- Python: upon success, a positive integer (the “handle”) uniquely identifying the axis, or if an error is encountered an exception is raised.

cmor_zfactor()

Fortran: zfactor_id = cmor_zfactor(zaxis_id, zfactor_name, [axis_ids], [units], zfactor_values, zfactor_bounds)

C: error_flag = cmor_zfactor (int *zfactor_id, int zaxis_id, char *zfactor_name, char *units, int ndims, int axis_ids[], char type, void *zfactor_values, void *zfactor_bounds)

Python: zfactor_id = zfactor(zaxis_id, zfactor_name, units, axis_ids, data_type, zfactor_values=None, zfactor_bounds=None)

*Description:* Define a factor needed to convert a non-dimensional vertical coordinate (model level) to a physical location. For pressure, height, or depth, this function is unnecessary, but for dimensionless coordinates it is needed. In the case of atmospheric sigma coordinates, for example, a scalar parameter must be defined indicating the top of the model, and the variable containing the surface pressure must be identified. The parameters that must be defined for different vertical dimensionless coordinates are listed in Appendix D of the CF convention document (http://www.cgd.ucar.edu/cms/eaton/cf-metadata). Often bounds for the zfactors will be needed (e.g., for hybrid sigma coordinates, “A’s” and “B’s” must be defined both for the layers and, often more importantly, for the layer interfaces). This function must be invoked for each z-factor required.

*Arguments*:

- **zfactor_id** = the “handle”: a positive integer returned by this function which uniquely identifies the grid defined in this call to CMOR and can subsequently be used in calls to CMOR.
- **zaxis_id** = an integer (“handle”) returned by cmor_axis (which must have been previously called) indicating which axis requires this factor.
- **zfactor_name** = name of the z-factor that will be defined by this function. This should correspond to an entry in the MIP table.
- **[axis_ids]** = an integer array containing the list of axis_id’s (individually defined by calls to cmor_axis), which the z-factor defined here is a function of (e.g. for surface pressure, the array of i.d.’s would usually include the longitude, latitude, and time axes.) The order of the axes must be consistent with the array passed as param_values. If the z-factor parameter is a function of a single dimension (e.g., model level), the single axis_id should be passed as an array of rank one and length 1, not as a scalar. If the parameter is a scalar, then this parameter may be omitted. If this parameter is carried on a non-cartesian latitude-longitude grid, then the grid_id should be passed instead of axis_ids, for latitude/longitude. Again if axis_ids collapses to a scalar, it should be passed as an array of rank one and length 1, not as a scalar.
- **[units]** = units associated with the z-factor passed in zfactor_values and zfactor_bounds. (These are the units of the user’s z-factors, which may differ from the units of the z-factors written to the netCDF file by CMOR.) . These units must be recognized by udunits or must be identical to the units specified in the MIP table. In the case of a dimensionless z-factors, either omit this argument, or set units=’’, or set units=’1’.
- **type** = type of the zfactor_values and zfactor_bounds (if present) passed to this function. This can be ‘d’ (double), ‘f’ (float), ‘l’ (long), ‘i’ (int), or ‘c’ (char).

- **[zfactor_values]** = z-factor values associated with dimensionless vertical coordinate identified by zaxis_id. If this z-factor is a function of time (e.g., surface pressure for sigma coordinates), the user can omit this argument and instead store the z-factor values by calling cmor_write. In that case the cmor_write argument, “var_id”, should be set to zfactor_id (returned by this function) and the argument, “store_with”, should be set to the variable id of the output field that requires zfactor as part of its metadata. When many fields are a function of the (dimensionless) model level, cmor_write will have to be called several times, with the same zfactor_id, but with different variable ids. If no values are passed, omit this argument.
- **[zfactor_bounds]** = z-factor values associated with the cell bounds of the vertical dimensionless coordinate. These values should be of the same type as the zfactor_values (e.g., if zfactor_values is double precision, then zfactor_bounds must also be double precision). If no bounds values are passed, omit this argument or set zfactor = ‘none’. This is a ONE dimensional array of length nlevs+1.

Returns:

- Fortran: a negative integer if an error is encountered; otherwise returns a positive integer (the “handle”) uniquely identifying the z-factor.
- C: 0 upon success.
- Python: upon success, a positive integer (the “handle”) uniquely identifying the z-factor, or if an error is encountered an exception is raised.

---

cmor_variable()

Fortran: var_id = cmor_variable([table], table_entry, units, axis_ids, [missing_value], [tolerance], [positive], [original_name], [history], [comment])

C: error_flag = int cmor_variable(int *var_id, char *table_entry, char *units, int ndims, int axis_ids[], char type, void *missing, double *tolerance, char *positive, char*original_name, char *history, char *comment)

Python: var_id = variable(table_entry, units, axis_ids, data_type=’f’, missing_value=None, tolerance = 1.e-4, positive=None, original_name=None, history=None, comment=None)

*Description:* Define a variable to be written by CMOR and indicate which axes are associated with it. This function prepares CMOR to write the file that will contain the data for this variable. This function returns a “handle” (var_id), uniquely identifying the variable, which will subsequently be passed as an argument to the cmor_write function. The variable specified by the table_entry argument must be found in the currently “set” CMOR table, as specified by the cmor_load_table and cmor_set_table functions, or as an option, it can be provided in the Fortran version (for backward compatibility) by the now deprecated “table” keyword argument. The cmor_variable function will typically be repeatedly invoked to define other variables. Note that backward compatibility was kept with the Fortran-only optional “table” keyword. But it is now recommended to use cmor_load_table and cmor_set_table instead (and necessary for C/Python).

*Arguments*:

- **var_id** = the “handle”: a positive integer returned by this function, which uniquely identifies the variable and can be used in subsequent calls to CMOR.

- **[table]** = character string containing the filename of the MIP-specific table where table_entry (described next) can be found (e.g., “CMIP5_table_amon”, ‘IPCC_table_A1’, ‘AMIP_table_1a’, ‘AMIP_table_2’, ‘CMIP_table_2’, etc.) In CMOR2 this is an optional argument and is deprecated because the table can be specified through the cmor_load_table and cmor_set_table functions.
- **table_entry** = name of the variable (as it appears in the MIP table) that this function defines.
- **units** = units of the data that will be passed to CMOR by function cmor_write. These units may differ from the units of the data output by CMOR. Whenever possible, this string should be interpretable by udunits (see http://my.unitdata.ucar.edu/content/software/udunits/). In the case of dimensionless quantities the units should be specified consistent with the CF conventions, so for example: percent, units=’%’; for a fraction, units=’1’; for parts per million, units=’1e-6’, etc.).
- **ndims** = number of axes the variable contains (i.e., the rank of the array), which in fact is the number of elements in the axis_ids array that will be processed by CMOR.
- **axis_ids** = 1-d array containing integers returned by cmor_axis, which specifies, via their “handles” (i.e., axis_ids), the axes associated with the variable that this function defines. These handles should be ordered consistently with the data that will be passed to CMOR through function cmor_write (see documentation below). If the size of the 1-d array is larger than the number of dimensions, the ‘unused’ dimension handles must be set to 0. Note that if the handle of a single axis is passed, it must not be passed as a scalar but as a rank 1 array of length 1. Scalar (“singleton”) dimensions defined in the MIP table may be omitted from axis_ids unless they have been explicitly redefined by the user through calls to cmor_axis. A “singleton” dimension that has been explicitly defined by the user should appear last in the list of axis_ids if the array of data passed to cmor_write for this variable actually omits this dimension; otherwise it should appear consistent with the position of the axis in the array of data passed to cmor_write. In the case of a non-Cartesian grid, replace the values of the grid specific axes (representing the lat/lon axes) with the single grid_id returned by cmor_grid.
- **type** = type of the missing_value, which must be the same as the type of the array that will be passed to cmor_write. The options are: ‘d’ (double), ‘f’ (float), ‘l’ (long) or ‘i’ (int).
- **[missing_value]** = scalar that is used to indicate missing data for this variable. It must be the same type as the data that will be passed to cmor_write. This missing_value will in general be replaced by a standard missing_value specified in the MIP table. If there are no missing data, and the user chooses not to declare the missing value, then this argument may be omitted or assigned the value ‘none’ (i.e., missing_value=’none’).
- **[tolerance]** = scalar (type real) indicating fractional tolerance allowed in missing values found in the data. A value will be considered missing if it lies within ±tolerance*missing_value of missing_value. The default tolerance for real and double precision missing values is 1.0e-4 and for integers 0. This argument is ignored if the missing_value argument is not present.
- **[positive]** = ‘up’ or ‘down’ depending on whether a user-passed vertical energy (heat) flux or surface momentum flux (stress) input to CMOR is positive when it is directed upward or downward, respectively. This information will be used by CMOR to determine whether a sign change is necessary to make the data consistent with the MIP requirements. This argument is required for vertical energy and salt fluxes, for “flux correction” fields, and for surface stress; it is ignored for all other variables.

- **[original_name]** = the name of the variable as it is commonly known at the user's home institute. If the variable passed to CMOR was computed in some simple way from two or more original fields (e.g., subtracting the upwelling and downwelling fluxes to get a net flux), then it is recommended that this be indicated in the "original_name" (e.g., "irup – irdown", where "irup" and "irdown" are the names of the original fields that were subtracted). If more complicated processing was required, this information would more naturally be included in a "history" attribute for this variable, described next.
- **[history]** = how the variable was processed before outputting through CMOR (e.g., give name(s) of the file(s) from which the data were read and indicate what calculations were performed, such as interpolating to standard pressure levels or adding 2 fluxes together). This information should allow someone at the user's institute to reproduce the procedure that created the CMOR output. Note that this history attribute is variable-specific, whereas the history attribute defined by cmor_dataset provides information concerning the model simulation itself or refers to processing procedures common to all variables (for example, mapping model output from an irregular grid to a Cartesian coordinate grid). Note that when appropriate, CMOR will also indicate in the "history" attribute any operations it performs on the data (e.g., scaling the data, changing the sign, changing its type, reordering the dimensions, reversing a coordinate's direction or offsetting longitude). Any user-defined history will precede the information generated by CMOR.
- **[comment]** = additional notes concerning this variable can be included here.

Returns:

- Fortran: a negative integer if an error is encountered; otherwise returns a positive integer (the "handle") uniquely identifying the variable.
- C: 0 upon success.
- Python: upon success, a positive integer (the "handle") uniquely identifying the variable, or if an error is encountered an exception is raised.

---

cmor_set_deflate()

Fortran: error_flag = cmor_set_deflate(var_id, shuffle, deflate, deflate_level)

C: error_flag = cmor_set_deflate(int var_id, int shuffle, int deflate, int deflate_level)

Python: set_deflate(var_id, shuffle, deflate, deflate_level)

*Description:* Sets netCDF4 shuffle and compression on a CMOR variable.

*Arguments*:

- **var_id** = the cmor variable id
- **shuffle** = if true, turn on netCDF the shuffle filter
- **deflate** = if true, turn on the deflate filter at the level specified by the deflate_level parameter
- **deflate_level** = if the deflate parameter is non-zero, deflate variable using value. Must be between 0 and 9

Returns upon success:

- Fortran: 0
- C: 0
- Python: 0

---

cmor_set_zstandard()

Fortran: error_flag = cmor_set_zstandard(var_id, zstandard_level)

C: error_flag = cmor_set_zstandard(int var_id, int zstandard_level)

Python: set_zstandard(var_id, zstandard_level)

*Description:* Sets netCDF4 Zstandard compression level on a CMOR variable. CMOR's deflate compression must be disabled (use cmor_set_deflate to set to false) to enable the use of Zstandard compression on the netCDF file's data variable.

*Arguments*:

- **var_id** = the cmor variable id
- **zstandard_level** = Compression level. Must be between -131072 and 22

Returns upon success:

- Fortran: 0
- C: 0
- Python: 0

---

cmor_set_quantize()

Fortran: error_flag = cmor_set_quantize(var_id, quantize_mode, quantize_nsd)

C: error_flag = cmor_set_quantize(int var_id, int quantize_mode, int quantize_nsd)

Python: set_quantize(var_id, quantize_mode, quantize_nsd)

*Description:* Sets netCDF4 quantization mode and number of significant digits/bits preserved.

*Arguments*:

- **var_id** = the cmor variable id
- **quantize_mode** = Quantization mode. Can be set to the following values.
  - 0: No quantization mode
  - 1: BitGroom
    - Determines the number of bits to retain for the user-requested number of significant decimal digits, and then sets excess bits to zero or one for alternate array values. This method is not recommended since it has poorer

compression ratios and more sharply changing least significant bits than Granular BitRound.

- 2: Granular BitRound **(recommended)**
  - Determines the number to retain for the user-requested number of significant decimal digits for each value in the array, and applies IEEE rounding to the least significant bits. This mode is preferred to BitGroom for retaining the user-specified number of decimal significant digits as it produces a better overall compression ratio and smoother least significant bits than BitGroom.
- 3: BitRound
  - The user directly specifies the number of significant bits to retain, and applies IEEE rounding to the least significant bits. This method is recommended for preserving the user-specified number of significant bits.

- **quantize_nsd** = Number of significant digits or bits. Its use is dependent on the quantization mode used.
  - 0 (No quantization mode): The value is ignored.
  - 1 (BitGroom) or 2 (Granular BitRound): The value can be set from 1 to 7 digits for floats or 1 to 23 digits for doubles.
  - 3 (BitRound): The value can be set from 1 to 15 bits for floats or 1 to 52 bits for doubles.

Returns upon success:

- Fortran: 0
- C: 0
- Python: 0

---

cmor_set_variable_attribute()

Fortran: error_flag = cmor_set_variable_attribute(integer var_id, character(*) name, character(*) type, character(*) value)

C: error_flag = cmor_set_variable_attribute(int variable_id, char *attribute_name, char type, void *value)

Python: set_variable_attribute(var_id,name,data_type,value)

*Description:* Defines an attribute to be associated with the variable specified by the variable_id. This function is unlikely to be called in preparing CMIP5 output, except to delete the “ext_cell_measures” attribute (setting it to a empty string). For this reason you can only set character type attributes at the moment via Python and Fortran.

*Arguments*:

- **variable_id** = the “handle” returned by cmor_variable (when the variable was defined), which will become better described by the attribute defined in this function.
- **attribute_name** = name of the attribute

- **type** = type of the attribute value passed, which can be ‘d’ (double), ‘f’ (float), ‘l’ (long), ‘i’ (int), or ‘c’ (char).
- **value** = whatever value you wish to set the attribute to (type defined by type argument).

Returns upon success:

- Fortran: 0
- C: 0
- Python: 0

---

cmor_get_variable_attribute()

Fortran: error_flag = cmor_get_variable_attribute(integer var_id, character(*) name, character *value)

C: error_flag = cmor_get_variable_attribute(int variable_id, char *attribute_name, char type, void *value)

Python: get_variable_attribute(var_id,name)

*Description:* retrieves an attribute value set for the variable specified by the variable_id. This function is unlikely to be called in preparing CMIP5 output. The Python and Fortran version will only work on attribute of character (string) type, otherwise chaotic results should be expected

*Arguments*:

- **variable_id** = the “handle” returned by cmor_variable (when the variable was defined) identifying which variable the attribute is associated with.
- **attribute_name** = name of the attribute
- **type** = type of the attribute value to be retrieved. This can be ‘d’ (double), ‘f’ (float), ‘l’ (long), ‘i’ (int), or ‘c’ (char)
- **value** = the argument that will accept the retrieved attribute.

Returns upon success:

- Fortran: 0
- C: 0
- Python: The attribute value

---

cmor_has_variable_attribute()

Fortran: error_flag = cmor_has_variable_attribute(integer var_id, character(*) name)

C: error_flag = cmor_has_variable_attribute(int variable_id, char *attribute_name)

Python: has_variable_attribute(var_id,name)

*Description:* Determines whether an attribute exists and is associated with the variable specified by variable_id, which is a handle returned to the user by a previous call to cmor_variable. This function is unlikely to be called in preparing CMIP5 output.

*Arguments*:

- **variable_id** = the “handle” specifying which variable is of interest. A variable_id is returned by cmor_variable each time a variable is defined.
- **attribute_name** = name of the attribute of interest.

Returns upon success (i.e., if the attribute is found):

- Fortran: 0
- C: 0
- Python: True

---

cmor_create_output_path()

Fortran: call cmor_create_output_path(var_id, path)

C : isfixed = cmor_create_output_path(int var_id, char *path)

Python: path = create_output_path(var_id)

*Description:* construct the output path, consistent with CMIP5 specifications, where the file will be stored.

*Arguments*:

- **var_id** = variable identification (as returned from cmor_variable) you wish to get the output path for.
- **path** = string (or pointer to a string), which is returned by the function and contains the output path.

Returns:

- Fortran: nothing it is a subroutine
- C: 0 upon success or 1 if the filed is a fixed field
- Python: the full path to the output file

---

cmor_write()

Fortran: error_flag = cmor_write(var_id, data, [file_suffix], [ntimes_passed], [time_vals], [time_bnds], [store_with])

C: error_flag = cmor_write(int var_id, void *data, char type, char *file_suffix, int ntimes_passed, double *time_vals, double *time_bounds, int *store_with)

Python: write(var_id, data, ntimes_passed=None, file_suffix=””, time_vals=None, time_bnds=None, store_with=None)

*Description:* For the variable identified by var_id, write an array of data that includes one or more time samples. This function will typically be repeatedly invoked to write other variables or append additional time samples of data. Note that time-slices of data must be written chronologically.

*Arguments*:

- **var_id** = integer returned by cmor_variable identifying the variable that will be written by this function.
- **data** = array of data written by this function (of rank<8). The rank of this array should either be: (a) consistent with the number of axes that were defined for it, or (b) it should be 1-dimensional, in which case the data must be stored contiguously in memory. In case (a), an exception is that for a variable that is a function of time and when only one "time-slice" is passed, then the array can optionally omit this dimension. Thus, for a variable that is a function of longitude, latitude, and time, for example, if only a single time-slice is passed to cmor_write, the rank of array "data" may be declared as either 2 or 3; when declared rank 3, the time-dimension will be size 1. It is recommended (but not required) that the shape of data (i.e., the size of each dimension) be consistent with those expected for this variable (based on the axis definitions), but they are allowed to be larger (the extra values beyond the defined dimension domain will be ignored). In any case the dimension sizes (lengths) must obviously not be smaller than those defined by the calls to cmor_axis.
- **type** = type of variable array ("data"), which can be 'd' (double), 'f' (float), 'l' (long) or 'i' (int).
- **[file_suffix]** = string that will be concatenated with a string automatically generated by CMOR to form a unique filename where the output is written. This suffix is only required when a time-sequence of output fields will not all be written into a single file (i.e., two or more files will contain the output for the variable). The file prefix generated by CMOR is of the form variable_table, where variable is replaced by table_entry (i.e., the name of the variable), and table is replaced by the table number (e.g., tas_A1 refers to surface air temperature as specified in table A1). Permitted characters will be: a-z, A-Z, 0-9, and "-". There are no restrictions on the suffix except that it must yield unique filenames and that it cannot contain any "*". If the user supplies a suffix, the leading* '' should be omitted (e.g., pass '1979-1988', not '_1979-1988'). Note that the suffix passed through cmor_write remains in effect for the particular variable until (optionally) redefined by a subsequent call. In the case of CMOR "Append mode" (in case the file already existed before a call to cmor_setup), then file_suffix is to be used to point to the original file, this value should reflect the FULL path where the file can be found, not just the file name. CMOR2 will be smart enough to figure out if a suffix was used when creating that file. Note that this file will be first moved to a temporary file and eventually renamed to reflect the additional times written to it.
- **[ntimes_passed]** = integer number of time slices passed on this call. If omitted, the number will be assumed to be the size of the time dimension of the data (if there is a time dimension).
- **[time_vals]** = 1-d array (must be double precision) time coordinate values associated with the data array. This argument should appear only if the time coordinate values were not passed in defining the time axis (i.e., in calling cmor_axis) such as when CMOR is set to "Append mode." The units should be consistent with those passed as an argument to cmor_axis in defining the time axis. If cell bounds are also passed (see next argument, '[time_bnds]'), then CMOR will first check that each coordinate value is not outside its associated cell bounds; subsequently, however, the user-defined coordinate value will be replaced by the mid-point of the interval defined by its bounds, and it is this value that will be written to the netCDF file.

- **[time_bnds]** = 2-d array (must be double precision) containing time bounds, which should be in the same units as time_vals. If the time_vals argument is omitted, this argument should also be omitted. The array should be dimensioned (2, n) in Fortran, and (n,2) in C/Python, where n is the size of time_vals (see CF standard document, http://www.cgd.ucar.edu/cms/eaton/cf-metadata, for further information).
- **[store_with]** = integer returned by cmor_variable identifying the variable that the zfactor should be stored with. This argument must be defined when and only when writing a z-factor. (See description of the zfactor function above.)

Returns upon success:

- Fortran: 0
- C: 0
- Python: None

---

cmor_close()

Fortran: error_flag = cmor_close(var_id, file_name, preserve)

C: error_flag = cmor_close(void) or

C: error_flag = cmor_close_variable(int var_id, char *file_name, int *preserve)

Python: error_flag (or if name=True, returns the name of the file) = close(var_id=None, file_name=False, preserve=False)

*Description:* Close a single file specified by optional argument var_id, or if this argument is omitted, close all files created by CMOR (including log files). To be safe, before exiting any program that invokes CMOR, it is best to call this function with the argument omitted. In C to close a single variable, use: cmor_close_variable(var_id). When using this function to close a single file, an additional optional argument (of type “string”) can be included, into which will be returned the file name created by CMOR. [In python, the string is returned by the function.] Another additional optional argument can be passed specifying if the variable should be preserved for future use (e.g., if you want to write additional data but to a new file). Note that when preserve is true, the original var_id is preserved.

*Arguments*:

- **[var_id]** = the “handle” identifying an individual variable and the associated output file that will be closed by this function.
- **[file_name]** = a string where the output file name will be stored. The file_name is returned only if its var_id has been included in the close_cmor argument list. This option provides a convenient method for the user to record the filename, which might be needed on a subsequent call to CMOR, for example, in order to append additional time samples to the file.
- **[preserve]** = Do you want to preserve the var definition? (0/1) If true, the original var_id is preserved.

Returns:

- Fortran: 0 upon success

- C: 0 upon success
- Python: None if file_name=False, or the name of the file if file_name=True and a var_id is passed as an argument.

---

cmor_get_terminate_signal()

Fortran: signal_code = cmor_get_terminate_signal()

C: signal_code = cmor_get_terminate_signal()

Python: signal_code = get_terminate_signal()

*Description:* Get the current value of the signal code issue by CMOR's C when encountering a termination error. Initially this is set to -999. If the user does not set it then the first call to cmor_setup will set the signal to SIGTERM for C and Python and to SIGINT for FORTRAN. FORTRAN does exit nicely with SIGTERM, hence the different default values

Returns upon success:

- Fortran: the current signal code
- C: the current signal code
- Python: the current signal code

---

cmor_set_terminate_signal()

Fortran: cmor_set_terminate_signal(signal)

C: cmor_set_terminate_signal(int signal)

Python: set_terminate_signal(signal)

*Description:* Set sthe signal code to send uppon termination from an error

*Arguments:*

- **[signal]** = an integer representing the signal code desired

---

# Acknowledgements

## Acknowledgements

Several individuals have supported the development of the CMOR1 software and provided encouragement, including Dean Williams, Dave Bader, and Peter Gleckler. Jonathan Gregory, Jim Boyle, and Bob Drach all provided valuable suggestions on how to simplify or in other ways improve the design of this software, and we particularly appreciate the time they spent reading and thinking about this problem. Jim Boyle additionally helped in a number of other ways, including porting CMOR to various platforms. Brian Eaton provided his usual careful and thoughtful responses to questions about CF compliance. Finally, we appreciate the encouragement expressed by the WGCM for developing CMOR.

The complete rewrite of CMOR, along with the new capabilities added to version 2, was implemented by Charles Doutriaux. We thank Dean Williams, Bob Drach, Renata McCoy, Jim Boyle, and the British Atmospheric Data Center (BADC). We also thank every one of the “early” adopters of CMOR2 who patiently helped us test and debug CMOR2. In particular we would like to thank Jamie Kettleborough from the UK Metoffice, Stephen Pascoe of the British Atmospheric Data Centre, Joerg Wegner of Zentrum für Marine und Atmosphärische Wissenschaften, Yana Malysheva of the Geophysical Fluid Dynamics Laboratory and Alejandro Bodas-Salcedo of UK Metoffice for the many lines of codes, bug fixes, and sample tests they sent our way

Enhancements to CMOR with capabilities added for version 3 were implemented by Denis Nadeau with help from Charles Doutriaux. We thanks Paul Durack and Martin Juckes who provided inputs, enhancement and solutions to improve flexibility. We also thank the “early” users of CMOR3 for their patience and for helping use improving CMOR3.



LLNL-CODE-2005936

# PrePARE

### Note

In order to use PrePARE please follow these instructions.

- Anaconda installation (https://cmor.llnl.gov/mydoc_cmor3_conda/)

PrePARE has been created to validate CMIP6 data before publishing files to ESGF. It may not work properly on CMIP5 files.

### Usage

```
PrePARE  [-h] [-l [CWD]] [--variable VARIABLE] [-v] [--table-path TABLE_PATH]
         [--max-processes 4] [--all] [--hide-progress] [--no-text-color]
         [--ignore-dir PYTHON_REGEX] [--include-file PYTHON_REGEX]
         [--exclude-file PYTHON_REGEX]
         input [input ...]
```

where:

- **input** Input CMIP6 netCDF data to validate. If a directory is submitted all netCDF recursively found will be validated independently.
- **-h** Display synopsis of the program.
- **-l, –log** Logfile directory. Default is the working directory. If not, standard output is used. Only available in multiprocessing mode.
- **–variable** Specify geophysical variable name. If not variable is deduced from filename.
- **-v, –version** Version of software.
- **–table-path** Specify the CMIP6 CMOR tables path (JSON file). If not submitted read the CMIP6_CMOR_TABLES environment variable if it exists. If a directory is submitted table is deduced from filename (default is “./Tables”).
    - CMIP6 tables (https://github.com/PCMDI/cmip6-cmor-tables/)
- **–max-processes** Maximum number of processes to simultaneously validate several files. Set to one seems sequential processing (default). Set to “-1” uses all available resources as returned by “multiprocessing.cpu_count()”.
- **–all** Show all results. Default only shows error(s) (i.e., file(s) not compliant).
- **–hide-progress** Do not show the percentage of progress / number of files checked while running PrePARE.
- **–no-text-color** Remove text color from output.

- **–ignore-dir** Filter directories NON-matching the regular expression. Default ignores paths with folder name(s) starting with ".".
- **–exclude-file** Filter files NON-matching the regular expression. Duplicate the flag to set several filters. Default only exclude hidden files (with names not starting with ".").

### Validation

PrePARE will verify that all attributes in the input file are present and conform to CMIP6 for publication into ESGF. We also recommand running the python program cfchecker (https://pypi.python.org/pypi/cfchecker) created by the University of Reading in the UK to confirm that your file is CF-1 compliant.

- In order to validate all CMIP6 required attributes by PrePARE, a Controlled Vocabulary file (https://github.com/PCMDI/cmip6-cmor-tables/blob/main/Tables/CMIP6_CV.json) is read by the program where a JSON dictionary called "required_global_attributes" (https://github.com/PCMDI/cmip6-cmor-tables/blob/a46dcbf17ec5e11af23dc2d55107f5e52afbcade/Tables/CMIP6_CV.json#L3)
  points to a list of strings. Each element of that list corresponds to a global attribute.
- PrePARE can also use regular expressions to validate the value of the some global attributes. Here is an example (https://github.com/PCMDI/cmip6-cmor-tables/blob/a46dcbf17ec5e11af23dc2d55107f5e52afbcade/Tables/CMIP6_CV.json#L11554-L11555)
  used for variant_label.
- Institutions and institution_ids need to be registered into a list. PrePARE will only accept institutions which have been pre-registered for CMIP6 publication into ESGF. Click here (https://github.com/PCMDI/cmip6-cmor-tables/blob/a46dcbf17ec5e11af23dc2d55107f5e52afbcade/Tables/CMIP6_CV.json#L74)
  for the list of institutions. If you wish to register your institution write to the cmor mailing list.
- Source and Source ID also need to be registered for CMIP6 publication. Here is the list (https://github.com/PCMDI/cmip6-cmor-tables/blob/a46dcbf17ec5e11af23dc2d55107f5e52afbcade/Tables/CMIP6_CV.json#L125)
  of registered sources.
- Only experiments found in the Controlled Vocabulary files are accepted for CMIP6 publication. A list of experiment_ids (https://github.com/PCMDI/cmip6-cmor-tables/blob/a46dcbf17ec5e11af23dc2d55107f5e52afbcade/Tables/CMIP6_CV.json#L3708)
  have been pre-defined including mandatory attributes. A warning will be displayed if one experiment attribute is missing or is not properly set by your program.
- grid and nominal_resolution are mandatory global attributes in CMIP6. PrePARE will make sure that these attributes are conformed to one of the following syntax:

| grid | nominal_resolution |
|---|---|
| gs1x1 | 1x1 degree |

| grid | nominal_resolution |
|---|---|
| gs1x1 or gn0 to gn9 | 1x1 degree |
| gs1x1 or gr0 to gr9 | 1x1 degree |
| gn0 to gn9 | “5 km” or “10 km” or “25 km” or “50 km” or “100 km” or “250 km” or “500 km” or “1000 km” or “2500 km” or “5000 km” or “10000 km” |
| gr0 to gr9 | “5 km” or “10 km” or “25 km” or “50 km” or “100 km” or “250 km” or “500 km” or “1000 km” or “2500 km” or “5000 km” or “10000 km” |

- PrePARE verifies that the creation date found in the netCDF file is conform to ISO 8601 (https://en.wikipedia.org/wiki/ISO_8601) standard.
- The Further Info URL attribute has to be set according to a very specific template. The PrePARE will use global attribute names found in the netCDF input file and replace the corresponding tag found in a template to rebuild proper CMIP6 link. If the reconstructed URL does not correspond to the value found in the input file, PrePARE will display an critical error on the screen.
    - Below is the defaul template used for the **furtherinfourl** attribute. Each string found between the “<>” characters correspond to a global attribute. The program will replace these strings with the corresponding global attribute values and add the **”.”** character between each tag.

```
http://furtherinfo.es-doc.org/<mip_era><institution_id><source_id><experiment_i
d><sub_experiment_id><variant_label>

becomes

http://furtherinfo.es-doc.org/CMIP6.CSIRO-BOM.NICAM.piControl.none.r1i1p1f1"
```

- PrePARE will also verify variable attributes necessary for CMIP6 publication. It validates: long_name, standard_name, units and missing_value according the CMIP6 tables information.

# Mamba installation

## Installing Miniforge

- **CMOR 3.10.0 on conda-forge has support for Python 3.9, 3.10, 3.11, 3.12, and 3.13.**
- Download the Miniforge installer (https://conda-forge.org/download/) for your system.
    - CMOR is currently only supported for Linux and macOS x86_64, and macOS arm64 (Apple Silicon)
- Start the install with the following command

```
bash Miniforge3-$(uname)-$(uname -m).sh
```

## Installing CMOR and PrePARE

- Run the following commands

```
# install cmor
# ------------------------------------------------
mamba create -n CMOR -c conda-forge cmor
mamba activate CMOR

# Clone the CMIP6 table to your working directory.
# ------------------------------------------------
mkdir CMIP6_work
cd  CMIP6_work
git clone https://github.com/PCMDI/cmip6-cmor-tables.git

# Note:
# ------------------------------------------------------------
# UDUNITS2_XML_PATH is set automatically by activating CMOR.
# export UDUNITS2_XML_PATH=${CONDA_PREFIX}/share/udunits/udunits2.xml
#
```

## Testing

- Run the full test suite for C, Fortran, and Python

  Clone CMOR code and CMIP6 tables

```
# Clone the CMOR repository to your working directory.
# ------------------------------------------------
git clone https://github.com/PCMDI/cmor.git
cd cmor

# Update the CMIP6 tables submodule.
# ------------------------------------------------
git submodule init
git submodule update
```

Install gcc and gfortran and linking environment variable

Linux:

```
mamba install -n CMOR -c conda-forge gcc_linux-64 gfortran_linux-64
export LDSHARED_FLAGS="-shared -pthread"
```

Mac:

```
mamba install -n CMOR -c conda-forge clang_osx-64 gfortran_osx-64
export LDSHARED_FLAGS=" -bundle -undefined dynamic_lookup"
```

Build and run tests

```
# Set prefix for configure step.
# ------------------------------------------------
export PREFIX=$(python -c "import sys; print(sys.prefix)")

# Configure the Makefile.
# ------------------------------------------------
./configure --prefix=$PREFIX --with-python --with-uuid=$PREFIX --with-jso
n-c=$PREFIX --with-udunits2=$PREFIX --with-netcdf=$PREFIX  --enable-verbo
se-test

# Run the tests with the Makefile (without rebuilding CMOR).
# ------------------------------------------------
make test -o cmor -o python
```

### Mamba environment

- Create your different CMOR environment with mamba.

```
mamba create -n [YOUR_ENV_NAME_HERE] -c conda-forge cmor
mamba activate [YOUR_ENV_NAME_HERE]
```

- To learn more about mamba environments (https://mamba.readthedocs.io/en/latest/user_guide/concepts.html)

### Obtaining Nighlty builds

- Create a dedicated environment for nightly (in between releases code):

```
mamba create -n [YOUR_ENV_NAME_HERE] -c pcmdi/label/nightly -c conda-forg
e cmor
mamba activate [YOUR_ENV_NAME_HERE]
```

- Create an environment with compilers for development/testing:

```
mamba create -n [YOUR_ENV_NAME_HERE] -c pcmdi/label/nightly -c conda-forg
e cmor gcc_linux-64 gfortran_linux-64
mamba activate [YOUR_ENV_NAME_HERE]
```

# Source installation

## Obtaining the CMOR and PrePARE source code and CMIP6 tables

- Clone the repo from gituhb

```
git clone git://github.com/pcmdi/cmor
cd cmor
git submodule init
git submodule update
```

## Installing Miniforge

- **CMOR 3.10.0 has support for Python 3.9, 3.10, 3.11, 3.12, and 3.13.**
- Download the Miniforge installer (https://conda-forge.org/download/) for your system.
    - CMOR is currently only supported for Linux and macOS x86_64, and macOS arm64 (Apple Silicon)
- Start the install with the following command

```
bash Miniforge3-$(uname)-$(uname -m).sh
```

### *Creating the mamba environement with compilers and needed libraries*

- Depending on your os mamba brings different compilers

  For Linux

```
export CONDA_COMPILERS="gcc_linux-64 gfortran_linux-64"
```

  For Mac

```
export CONDA_COMPILERS="clang_osx-64 gfortran_osx-64"
```

- Run the following command to build CMOR for your version of Python

  Python 3.11

```
mamba create -n cmor_dev -c conda-forge six libuuid json-c udunits2 hdf5
libnetcdf openblas netcdf4 numpy openssl python=3.11 $CONDA_COMPILERS
```

- Activate the mamba environment

```
mamba activate cmor_dev
```

## Configuring cmor

- Depending on your OS linking environment variables are different

  For Linux

```
export LDSHARED_FLAGS="-shared -pthread"
```

  For Mac

```
export LDSHARED_FLAGS=" -bundle -undefined dynamic_lookup"
```

- Set the PREFIX

  Since your environment can use a different name and its location is system dependent use:

```
export PREFIX=$(python -c "import sys; print(sys.prefix)")
```

- configure cmor:

```
./configure --prefix=$PREFIX --with-python --with-uuid=$PREFIX --with-jso
n-c=$PREFIX --with-udunits2=$PREFIX --with-netcdf=$PREFIX  --enable-verbo
se-test
```

## Building and installing CMOR and PrePARE

- Run

```
make install
```

### Testing the installation

- Run C, Fortran, and Python tests

```
make test
```

# Example Python

## CMOR Input Files

- CMOR_input_example.json (https://github.com/PCMDI/cmor/blob/main/Test/CMOR_input_example.json)
- CMIP6_coordinate.json (https://github.com/PCMDI/cmip6-cmor-tables/blob/main/Tables/CMIP6_coordinate.json)
- CMIP6_formula_terms.json (https://github.com/PCMDI/cmip6-cmor-tables/blob/main/Tables/CMIP6_formula_terms.json)
- CMIP6_CV.json (https://github.com/PCMDI/cmip6-cmor-tables/blob/main/Tables/CMIP6_CV.json)
- CMIP6_Amon.json (https://github.com/PCMDI/cmip6-cmor-tables/blob/main/Tables/CMIP6_Amon.json)
- CMIP6_Omon.json (https://github.com/PCMDI/cmip6-cmor-tables/blob/main/Tables/CMIP6_Omon.json)

## Example 1: Python source code

- test_doc.py (https://github.com/PCMDI/cmor/blob/main/Test/test_doc.py)

**Click to expand Python code**

```
import cmor

cmor.setup(
        # inpath has to point to the CMOR
        # tables path (CMIP6, input4MIPs or otherwise)
        inpath='Tables',
        netcdf_file_action=cmor.CMOR_REPLACE_4
)

cmor.dataset_json("Test/CMOR_input_example.json")

# Loading this test table overwrites the normal CF checks on valid variable valu
es.
# This is perfect for testing but shouldn't be done when writing real data.
table='CMIP6_Amon.json'
cmor.load_table(table)


# here is where you add your axes
itime = cmor.axis(table_entry= 'time',
                  units= 'days since 2000-01-01 00:00:00',
                  coord_vals= [15,],
                  cell_bounds= [0, 30])
ilat = cmor.axis(table_entry= 'latitude',
                 units= 'degrees_north',
                 coord_vals= [0],
                 cell_bounds= [-1, 1])
ilon = cmor.axis(table_entry= 'longitude',
                 units= 'degrees_east',
                 coord_vals= [90],
                 cell_bounds= [89, 91])

axis_ids = [itime,ilat,ilon]

# here we create a variable with appropriate name, units and axes
varid = cmor.variable('ts', 'K', axis_ids)

# then we can write the variable along with the data
cmor.write(varid, [273])

# finally we close the file and print where it was saved
outfile = cmor.close(varid, file_name=True)
print("File written to: {}".format(outfile))
cmor.close()
```

## Example 2: Usual Treatment of a 2-D Field

- example2.py (page 0)

**click to expand Python code**

```
import cmor
import numpy
import os

hfls = numpy.array([120, 116, 112, 108,
                    104, 100, 96, 92,
                    88, 84, 80, 76,
                    119, 115, 111, 107,
                    103, 99, 95, 91,
                    87, 83, 79, 75
                    ])
hfls.shape = (2, 3, 4)
lat = numpy.array([10, 20, 30])
lat_bnds = numpy.array([5, 15, 25, 35])
lon = numpy.array([0, 90, 180, 270])
lon_bnds = numpy.array([-45, 45,
                        135,
                        225,
                        315
                        ])
time = numpy.array([15.5, 45])
time_bnds = numpy.array([0, 31, 60])
ipth = opth = 'test'
cmor.setup(inpath=ipth,
           set_verbosity=cmor.cmor_normal,
           netcdf_file_action=cmor.cmor_replace)
cmor.dataset_json("cmor_input_example.json")
cmor.load_table("cmip6_amon.json")
cmorlat = cmor.axis("latitude",
                    coord_vals=lat,
                    cell_bounds=lat_bnds,
                    units="degrees_north")
cmorlon = cmor.axis("longitude",
                    coord_vals=lon,
                    cell_bounds=lon_bnds,
                    units="degrees_east")
cmortime = cmor.axis("time",
                     coord_vals=time,
                     cell_bounds=time_bnds,
                     units="days since 2018")
axes = [cmortime, cmorlat, cmorlon]
cmorhfls = cmor.variable("hfls", "w/m2", axes, positive="up")
cmor.write(cmorhfls, hfls)
filename = cmor.close(cmorhfls, file_name=true)
print("stored in:", filename)
cmor.close()
```

**click to expand netcdf dump**

```
netcdf hfls_amon_pcmdi-test-1-0_picontrol-withism_r3i1p1f1_gn_201801-201802 {
dimensions:
        time = unlimited ; // (2 currently)
        lat = 3 ;
        lon = 4 ;
        bnds = 2 ;
variables:
        double time(time) ;
                time:bounds = "time_bnds" ;
                time:units = "days since 2018" ;
                time:calendar = "360_day" ;
                time:axis = "t" ;
                time:long_name = "time" ;
                time:standard_name = "time" ;
        double time_bnds(time, bnds) ;
        double lat(lat) ;
                lat:bounds = "lat_bnds" ;
                lat:units = "degrees_north" ;
                lat:axis = "y" ;
                lat:long_name = "latitude" ;
                lat:standard_name = "latitude" ;
        double lat_bnds(lat, bnds) ;
        double lon(lon) ;
                lon:bounds = "lon_bnds" ;
                lon:units = "degrees_east" ;
                lon:axis = "x" ;
                lon:long_name = "longitude" ;
                lon:standard_name = "longitude" ;
        double lon_bnds(lon, bnds) ;
        float hfls(time, lat, lon) ;
                hfls:standard_name = "surface_upward_latent_heat_flux" ;
                hfls:long_name = "surface upward latent heat flux" ;
                hfls:comment = "the surface called \'surface\' means the lower b
oundary of the atmosphere. \'upward\' indicates a vector component which is posi
tive when directed upward (negative downward). the surface latent heat flux is t
he exchange of heat between the surface and the air on account of evaporation (i
ncluding sublimation). in accordance with common usage in geophysical discipline
s, \'flux\' implies per unit area, called \'flux density\' in physics." ;
                hfls:units = "w m-2" ;
                hfls:original_units = "w/m2" ;
                hfls:history = "2019-01-08t23:32:26z altered by cmor: converted
units from \'w/m2\' to \'w m-2\'. 2019-01-08t23:32:26z altered by cmor: converte
d type from \'l\' to \'f\'." ;
                hfls:cell_methods = "area: time: mean" ;
                hfls:cell_measures = "area: areacella" ;
                hfls:missing_value = 1.e+20f ;
                hfls:_fillvalue = 1.e+20f ;

// global attributes:
                :conventions = "cf-1.7 cmip-6.2" ;
```

```
                :activity_id = "ismip6" ;
                :branch_method = "no parent" ;
                :branch_time_in_child = 59400. ;
                :branch_time_in_parent = 0. ;
                :contact = "python coder (coder@a.b.c.com)" ;
                :creation_date = "2019-01-08t23:32:26z" ;
                :data_specs_version = "01.00.27" ;
                :experiment = "preindustrial control with interactive ice shee
t" ;
                :experiment_id = "picontrol-withism" ;
                :external_variables = "areacella" ;
                :forcing_index = 1 ;
                :frequency = "mon" ;
                :further_info_url = "https://furtherinfo.es-doc.org/cmip6.pcmd
i.pcmdi-test-1-0.picontrol-withism.none.r3i1p1f1" ;
                :grid = "native atmosphere regular grid (3x4 latxlon)" ;
                :grid_label = "gn" ;
                :history = "2019-01-08t23:32:26z ;rewrote data to be consistent
with ismip6 for variable hfls found in table amon.;\n",
                        "output from archivcl_a1.nce/giccm_03_std_2xco2_2256." ;
                :initialization_index = 1 ;
                :institution = "program for climate model diagnosis and intercom
parison, lawrence livermore national laboratory, livermore, ca 94550, usa" ;
                :institution_id = "pcmdi" ;
                :mip_era = "cmip6" ;
                :nominal_resolution = "10000 km" ;
                :parent_activity_id = "no parent" ;
                :parent_experiment_id = "no parent" ;
                :parent_mip_era = "no parent" ;
                :parent_source_id = "no parent" ;
                :parent_time_units = "no parent" ;
                :parent_variant_label = "no parent" ;
                :physics_index = 1 ;
                :product = "model-output" ;
                :realization_index = 3 ;
                :realm = "atmos" ;
                :references = "model described by koder and tolkien (j. geophy
s. res., 2001, 576-591).  also see http://www.gicc.su/giccm/doc/index.html.  th
e ssp245 simulation is described in dorkey et al. \'(clim. dyn., 2003, 323-35
7.)\'" ;
                :run_variant = "3rd realization" ;
                :source = "pcmdi-test 1.0 (1989): \n",
                        "aerosol: none\n",
                        "atmos: earth1.0-gettinghotter (360 x 180 longitude/lati
tude; 50 levels; top level 0.1 mb)\n",
                        "atmoschem: none\n",
                        "land: earth1.0\n",
                        "landice: none\n",
                        "ocean: bluemarble1.0-warming (360 x 180 longitude/latit
ude; 50 levels; top grid cell 0-10 m)\n",
```

```
                        "ocnbgchem: none\n",
                        "seaice: declining1.0-warming (360 x 180 longitude/latit
ude)" ;
                :source_id = "pcmdi-test-1-0" ;
                :source_type = "aogcm ism aer" ;
                :sub_experiment = "none" ;
                :sub_experiment_id = "none" ;
                :table_id = "amon" ;
                :table_info = "creation date:(30 july 2018) md5:fa9bc503f57fb067
bf398cab2c4ba77e" ;
                :title = "pcmdi-test-1-0 output prepared for cmip6" ;
                :tracking_id = "hdl:21.14100/ded65b61-6588-48f6-bd07-7e4281be9be
e" ;
                :variable_id = "hfls" ;
                :variant_label = "r3i1p1f1" ;
                :license = "cmip6 model data produced by lawrence livermore pcmd
i is licensed under a creative commons attribution sharealike 4.0 international
license (https://creativecommons.org/licenses). consult https://pcmdi.llnl.gov/c
mip6/termsofuse for terms of use governing cmip6 output, including citation requ
irements and proper acknowledgment. further information about this data, includi
ng some limitations, can be found via the further_info_url (recorded as a globa
l attribute in this file) and at https:///pcmdi.llnl.gov/. the data producers an
d data providers make no warranty, either express or implied, including, but no
t limited to, warranties of merchantability and fitness for a particular purpos
e. all liabilities arising from the supply of the information (including any lia
bility arising in negligence) are excluded to the fullest extent permitted by la
w." ;
                :cmor_version = "3.4.0" ;
data:

 time = 15.5, 45.5 ;

 time_bnds =
  0, 31,
  31, 60 ;

 lat = 10, 20, 30 ;

 lat_bnds =
  5, 15,
  15, 25,
  25, 35 ;

 lon = 0, 90, 180, 270 ;

 lon_bnds =
  -45, 45,
  45, 135,
  135, 225,
  225, 315 ;
```

```
 hfls =
  120, 116, 112, 108,
  104, 100, 96, 92,
  88, 84, 80, 76,
  119, 115, 111, 107,
  103, 99, 95, 91,
  87, 83, 79, 75 ;
}
```

### Example 3: Usual Treatment of a 3-D Field on Pressure Levels

- example3.py (page 0)

**Click to expand Python code**

```
import cmor
import numpy
import os

ta = 10. * numpy.random.random_sample((2, 19, 3, 4)) + 250.
lat = numpy.array([10, 20, 30])
lat_bnds = numpy.array([5, 15, 25, 35])
lon = numpy.array([0, 90, 180, 270])
lon_bnds = numpy.array([-45, 45,
                        135,
                        225,
                        315
                        ])
time = numpy.array([15.5, 45])
time_bnds = numpy.array([0, 31, 60])
lev = numpy.array([100000, 92500, 85000, 70000, 60000, 50000, 40000, 30000,
                   25000, 20000, 15000, 10000, 7000, 5000, 3000, 2000, 1000, 50
0, 100])
ipth = opth = 'Test'
cmor.setup(inpath=ipth,
           set_verbosity=cmor.CMOR_NORMAL,
           netcdf_file_action=cmor.CMOR_REPLACE)
cmor.dataset_json("CMOR_input_example.json")
cmor.load_table("CMIP6_Amon.json")
cmorLat = cmor.axis("latitude",
                    coord_vals=lat,
                    cell_bounds=lat_bnds,
                    units="degrees_north")
cmorLon = cmor.axis("longitude",
                    coord_vals=lon,
                    cell_bounds=lon_bnds,
                    units="degrees_east")
cmorTime = cmor.axis("time",
                     coord_vals=time,
                     cell_bounds=time_bnds,
                     units="days since 2018")
cmorLev = cmor.axis("plev19", coord_vals=lev, units='Pa')
axes = [cmorTime, cmorLev, cmorLat, cmorLon]
cmorTa = cmor.variable("ta", "K", axes)
cmor.write(cmorTa, ta)
filename = cmor.close(cmorTa, file_name=True)
print("Stored in:", filename)
cmor.close()
os.system("ncdump {}".format(filename))
```

**Click to expand NetCDF dump**

```
netcdf ta_Amon_PCMDI-test-1-0_piControl-withism_r3i1p1f1_gn_201801-201802 {
dimensions:
        time = UNLIMITED ; // (2 currently)
        plev = 19 ;
        lat = 3 ;
        lon = 4 ;
        bnds = 2 ;
variables:
        double time(time) ;
                time:bounds = "time_bnds" ;
                time:units = "days since 2018" ;
                time:calendar = "360_day" ;
                time:axis = "T" ;
                time:long_name = "time" ;
                time:standard_name = "time" ;
        double time_bnds(time, bnds) ;
        double plev(plev) ;
                plev:units = "Pa" ;
                plev:axis = "Z" ;
                plev:positive = "down" ;
                plev:long_name = "pressure" ;
                plev:standard_name = "air_pressure" ;
        double lat(lat) ;
                lat:bounds = "lat_bnds" ;
                lat:units = "degrees_north" ;
                lat:axis = "Y" ;
                lat:long_name = "latitude" ;
                lat:standard_name = "latitude" ;
        double lat_bnds(lat, bnds) ;
        double lon(lon) ;
                lon:bounds = "lon_bnds" ;
                lon:units = "degrees_east" ;
                lon:axis = "X" ;
                lon:long_name = "Longitude" ;
                lon:standard_name = "longitude" ;
        double lon_bnds(lon, bnds) ;
        float ta(time, plev, lat, lon) ;
                ta:standard_name = "air_temperature" ;
                ta:long_name = "Air Temperature" ;
                ta:comment = "Air Temperature" ;
                ta:units = "K" ;
                ta:cell_methods = "time: mean" ;
                ta:cell_measures = "area: areacella" ;
                ta:missing_value = 1.e+20f ;
                ta:_FillValue = 1.e+20f ;
                ta:history = "2019-01-08T23:35:44Z altered by CMOR: Converted ty
pe from \'d\' to \'f\'." ;

// global attributes:
                :Conventions = "CF-1.7 CMIP-6.2" ;
```

```
                :activity_id = "ISMIP6" ;
                :branch_method = "no parent" ;
                :branch_time_in_child = 59400. ;
                :branch_time_in_parent = 0. ;
                :contact = "Python Coder (coder@a.b.c.com)" ;
                :creation_date = "2019-01-08T23:35:44Z" ;
                :data_specs_version = "01.00.27" ;
                :experiment = "preindustrial control with interactive ice shee
t" ;
                :experiment_id = "piControl-withism" ;
                :external_variables = "areacella" ;
                :forcing_index = 1 ;
                :frequency = "mon" ;
                :further_info_url = "https://furtherinfo.es-doc.org/CMIP6.PCMD
I.PCMDI-test-1-0.piControl-withism.none.r3i1p1f1" ;
                :grid = "native atmosphere regular grid (3x4 latxlon)" ;
                :grid_label = "gn" ;
                :history = "2019-01-08T23:35:44Z ;rewrote data to be consistent
with ISMIP6 for variable ta found in table Amon.;\n",
                        "Output from archivcl_A1.nce/giccm_03_std_2xCO2_2256." ;
                :initialization_index = 1 ;
                :institution = "Program for Climate Model Diagnosis and Intercom
parison, Lawrence Livermore National Laboratory, Livermore, CA 94550, USA" ;
                :institution_id = "PCMDI" ;
                :mip_era = "CMIP6" ;
                :nominal_resolution = "10000 km" ;
                :parent_activity_id = "no parent" ;
                :parent_experiment_id = "no parent" ;
                :parent_mip_era = "no parent" ;
                :parent_source_id = "no parent" ;
                :parent_time_units = "no parent" ;
                :parent_variant_label = "no parent" ;
                :physics_index = 1 ;
                :product = "model-output" ;
                :realization_index = 3 ;
                :realm = "atmos" ;
                :references = "Model described by Koder and Tolkien (J. Geophy
s. Res., 2001, 576-591).  Also see http://www.GICC.su/giccm/doc/index.html.  Th
e ssp245 simulation is described in Dorkey et al. \'(Clim. Dyn., 2003, 323-35
7.)\'" ;
                :run_variant = "3rd realization" ;
                :source = "PCMDI-test 1.0 (1989): \n",
                        "aerosol: none\n",
                        "atmos: Earth1.0-gettingHotter (360 x 180 longitude/lati
tude; 50 levels; top level 0.1 mb)\n",
                        "atmosChem: none\n",
                        "land: Earth1.0\n",
                        "landIce: none\n",
                        "ocean: BlueMarble1.0-warming (360 x 180 longitude/latit
ude; 50 levels; top grid cell 0-10 m)\n",
```

```
                        "ocnBgchem: none\n",
                        "seaIce: Declining1.0-warming (360 x 180 longitude/latit
ude)" ;
                :source_id = "PCMDI-test-1-0" ;
                :source_type = "AOGCM ISM AER" ;
                :sub_experiment = "none" ;
                :sub_experiment_id = "none" ;
                :table_id = "Amon" ;
                :table_info = "Creation Date:(30 July 2018) MD5:fa9bc503f57fb067
bf398cab2c4ba77e" ;
                :title = "PCMDI-test-1-0 output prepared for CMIP6" ;
                :tracking_id = "hdl:21.14100/11fd8d79-f1d9-453d-8293-7603dc5dfe1
e" ;
                :variable_id = "ta" ;
                :variant_label = "r3i1p1f1" ;
                :license = "CMIP6 model data produced by Lawrence Livermore PCMD
I is licensed under a Creative Commons Attribution ShareAlike 4.0 International
License (https://creativecommons.org/licenses). Consult https://pcmdi.llnl.gov/C
MIP6/TermsOfUse for terms of use governing CMIP6 output, including citation requ
irements and proper acknowledgment. Further information about this data, includi
ng some limitations, can be found via the further_info_url (recorded as a globa
l attribute in this file) and at https:///pcmdi.llnl.gov/. The data producers an
d data providers make no warranty, either express or implied, including, but no
t limited to, warranties of merchantability and fitness for a particular purpos
e. All liabilities arising from the supply of the information (including any lia
bility arising in negligence) are excluded to the fullest extent permitted by la
w." ;
                :cmor_version = "3.4.0" ;
data:

 time = 15.5, 45.5 ;

 time_bnds =
  0, 31,
  31, 60 ;

 plev = 100000, 92500, 85000, 70000, 60000, 50000, 40000, 30000, 25000,
    20000, 15000, 10000, 7000, 5000, 3000, 2000, 1000, 500, 100 ;

 lat = 10, 20, 30 ;

 lat_bnds =
  5, 15,
  15, 25,
  25, 35 ;

 lon = 0, 90, 180, 270 ;

 lon_bnds =
  -45, 45,
```

```
 45, 135,
 135, 225,
 225, 315 ;

ta =
 259.8408, 258.941, 252.4908, 251.0074,
 256.835, 258.2486, 250.7763, 256.6857,
 255.0459, 250.3535, 255.2871, 259.8668,
 253.4483, 256.5141, 259.7679, 252.1754,
 253.76, 250.867, 251.4578, 254.0015,
 252.3708, 258.2815, 255.3655, 257.9578,
 250.4145, 256.8469, 252.7675, 255.7654,
 258.5681, 259.7048, 254.8929, 252.6632,
 253.612, 258.3735, 256.299, 256.6488,
 253.1254, 254.9136, 255.2808, 253.8569,
 257.0341, 251.0754, 254.2664, 252.9441,
 255.2702, 255.9075, 253.3035, 257.6259,
 255.9043, 252.595, 253.9338, 258.0882,
 256.6787, 253.8391, 258.4736, 256.7391,
 252.671, 258.3957, 252.4797, 253.4284,
 258.6154, 258.6605, 255.7659, 252.4936,
 256.9681, 254.4894, 252.7608, 254.783,
 250.0667, 255.6354, 258.6563, 259.6775,
 254.3134, 250.8413, 250.2444, 254.6606,
 255.1599, 257.21, 252.6211, 256.2644,
 254.1385, 257.6524, 252.9917, 251.8146,
 259.5888, 256.2946, 255.9592, 254.3341,
 258.4659, 259.7941, 258.0905, 258.1966,
 257.4232, 259.0263, 251.2242, 259.6272,
 257.456, 259.999, 258.7972, 256.6927,
 256.5442, 250.3652, 254.9976, 254.2698,
 256.8535, 252.3693, 259.0882, 258.8849,
 253.6198, 259.3818, 251.4028, 254.9819,
 250.2339, 251.6083, 256.1446, 258.8874,
 259.977, 252.5453, 253.6287, 252.0314,
 257.0711, 254.6262, 253.957, 258.0355,
 250.333, 255.8116, 252.4928, 250.0648,
 256.737, 251.6935, 251.2943, 250.1732,
 256.9901, 256.2216, 259.1346, 258.9967,
 258.4938, 257.6587, 252.6762, 256.5123,
 254.6203, 257.5838, 259.8249, 257.2421,
 251.5132, 250.6997, 255.1321, 255.9642,
 250.5069, 251.3887, 252.1133, 253.0272,
 257.2178, 251.7756, 256.3568, 255.6339,
 250.4361, 258.3318, 259.5203, 258.9857,
 254.0756, 251.2124, 252.6628, 259.1253,
 252.2022, 254.2113, 259.4847, 252.9702,
 251.1179, 259.7756, 253.7968, 257.081,
 254.7999, 253.5379, 259.9748, 257.6128,
 257.1583, 258.5191, 252.2605, 251.6866,
```

```
251.9091, 255.4374, 259.1645, 255.4471,
251.6325, 252.1992, 256.1027, 252.4458,
251.5014, 252.293, 250.0457, 251.2812,
252.3479, 253.4959, 250.742, 254.3526,
254.2659, 258.3052, 259.6293, 253.8284,
255.0674, 250.3642, 252.008, 258.3384,
258.2568, 257.897, 253.424, 254.922,
254.939, 253.0223, 257.8987, 256.4419,
254.1967, 257.1554, 253.706, 256.5611,
259.3974, 254.3611, 251.2371, 257.0125,
255.4547, 255.4249, 252.8776, 257.173,
258.8824, 252.4057, 250.6023, 253.0139,
255.5045, 257.2598, 257.7797, 253.2221,
253.2658, 252.578, 255.4973, 252.6226,
259.9648, 251.6002, 254.1322, 251.8064,
254.0982, 252.5025, 251.4612, 251.3052,
254.25, 251.7179, 251.4255, 256.1079,
257.282, 255.7924, 252.2107, 255.8521,
252.7759, 253.0962, 252.638, 255.9666,
259.5663, 253.6493, 256.8842, 255.4041,
254.6592, 255.4181, 258.7055, 254.7371,
257.1625, 255.0113, 253.0983, 252.3584,
257.2872, 253.2124, 256.9593, 250.7623,
257.287, 257.2986, 252.6907, 251.519,
254.1635, 250.1762, 256.4446, 259.7633,
259.0938, 253.0846, 250.5819, 258.3493,
258.6836, 257.3046, 255.8258, 257.9142,
257.3879, 252.5779, 255.3217, 254.2868,
255.9011, 252.4209, 253.7516, 255.2315,
257.6159, 257.8478, 253.0855, 254.9258,
255.5278, 257.5585, 258.8186, 250.2102,
250.1217, 256.8153, 255.6852, 255.1126,
257.7202, 256.3726, 252.1865, 253.0071,
251.8633, 255.1148, 254.0191, 253.4735,
257.7912, 253.4579, 255.0269, 250.1707,
253.9734, 254.2229, 257.9263, 250.1125,
258.5282, 250.8859, 257.5581, 255.8632,
252.3537, 253.3955, 252.4796, 251.3484,
259.0306, 250.2142, 253.429, 251.3313,
257.0627, 253.5232, 254.0599, 253.5528,
254.7048, 257.3163, 258.5922, 259.0777,
253.2622, 258.1998, 254.3777, 259.3747,
252.3826, 251.1805, 253.8417, 251.8928,
256.3958, 256.0284, 253.0213, 256.2551,
258.8141, 258.835, 259.7631, 254.7228,
254.3701, 251.3905, 250.6818, 258.296,
258.9814, 258.1483, 256.1503, 252.6487,
252.123, 256.3247, 252.2733, 259.7609,
259.3987, 252.3202, 250.5132, 252.3688,
257.4306, 256.0952, 253.014, 251.8331,
```

```
  253.5333, 259.0857, 257.7149, 259.9082,
  259.6393, 258.5578, 256.9663, 252.2192,
  254.2118, 251.6638, 255.6581, 255.7678,
  253.8299, 251.3065, 255.6969, 259.1021,
  256.2309, 257.8936, 251.7329, 253.7878,
  256.5732, 254.0137, 253.6299, 250.413,
  258.7727, 251.4784, 253.509, 255.0283,
  254.1883, 255.0535, 250.6044, 257.4061,
  252.835, 255.2766, 257.4215, 259.0279,
  255.3313, 254.1923, 254.1777, 258.1096,
  250.3586, 255.7441, 258.2351, 257.1729,
  258.3901, 256.7424, 259.8389, 254.8441,
  252.9138, 256.6953, 255.918, 253.6417,
  252.3907, 255.4751, 258.3704, 255.8665,
  250.0418, 251.8351, 258.5436, 256.8799,
  252.2263, 255.383, 253.3702, 253.7597,
  251.112, 254.6407, 251.4067, 252.6422,
  258.1817, 252.3663, 253.5502, 252.4341,
  257.1426, 254.4529, 254.8314, 254.7638 ;
}
```

### Example 4: Treatment of a Scalar Dimension (near-surface air temperature)

- example4.py (page 0)

**Click to expand Python code**

```
import cmor
import numpy
import os

tas = 10. * numpy.random.random_sample((2, 3, 4)) + 250.
lat = numpy.array([10, 20, 30])
lat_bnds = numpy.array([5, 15, 25, 35])
lon = numpy.array([0, 90, 180, 270])
lon_bnds = numpy.array([-45, 45,
                        135,
                        225,
                        315
                        ])
time = numpy.array([15.5, 45])
time_bnds = numpy.array([0, 31, 60])
ipth = opth = 'Test'
cmor.setup(inpath=ipth,
           set_verbosity=cmor.CMOR_NORMAL,
           netcdf_file_action=cmor.CMOR_REPLACE)
cmor.dataset_json("CMOR_input_example.json")
cmor.load_table("CMIP6_Amon.json")
cmorLat = cmor.axis("latitude",
                    coord_vals=lat,
                    cell_bounds=lat_bnds,
                    units="degrees_north")
cmorLon = cmor.axis("longitude",
                    coord_vals=lon,
                    cell_bounds=lon_bnds,
                    units="degrees_east")
cmorTime = cmor.axis("time",
                     coord_vals=time,
                     cell_bounds=time_bnds,
                     units="days since 2018")
axes = [cmorTime, cmorLat, cmorLon]
cmorTas = cmor.variable("tas", "K", axes)
cmor.write(cmorTas, tas)
filename = cmor.close(cmorTas, file_name=True)
print("Stored in:", filename)
cmor.close()
```

**Click to expand NetCDF dump**

```
netcdf tas_Amon_PCMDI-test-1-0_piControl-withism_r3i1p1f1_gn_201801-201802 {
dimensions:
        time = UNLIMITED ; // (2 currently)
        lat = 3 ;
        lon = 4 ;
        bnds = 2 ;
variables:
        double time(time) ;
                time:bounds = "time_bnds" ;
                time:units = "days since 2018" ;
                time:calendar = "360_day" ;
                time:axis = "T" ;
                time:long_name = "time" ;
                time:standard_name = "time" ;
        double time_bnds(time, bnds) ;
        double lat(lat) ;
                lat:bounds = "lat_bnds" ;
                lat:units = "degrees_north" ;
                lat:axis = "Y" ;
                lat:long_name = "latitude" ;
                lat:standard_name = "latitude" ;
        double lat_bnds(lat, bnds) ;
        double lon(lon) ;
                lon:bounds = "lon_bnds" ;
                lon:units = "degrees_east" ;
                lon:axis = "X" ;
                lon:long_name = "Longitude" ;
                lon:standard_name = "longitude" ;
        double lon_bnds(lon, bnds) ;
        double height ;
                height:units = "m" ;
                height:axis = "Z" ;
                height:positive = "up" ;
                height:long_name = "height" ;
                height:standard_name = "height" ;
        float tas(time, lat, lon) ;
                tas:standard_name = "air_temperature" ;
                tas:long_name = "Near-Surface Air Temperature" ;
                tas:comment = "near-surface (usually, 2 meter) air temperature"
;
                tas:units = "K" ;
                tas:cell_methods = "area: time: mean" ;
                tas:cell_measures = "area: areacella" ;
                tas:history = "2019-01-08T23:41:05Z altered by CMOR: Treated sca
lar dimension: \'height\'. 2019-01-08T23:41:05Z altered by CMOR: Converted type
from \'d\' to \'f\'." ;
                tas:coordinates = "height" ;
                tas:missing_value = 1.e+20f ;
                tas:_FillValue = 1.e+20f ;
```

```
// global attributes:
                :Conventions = "CF-1.7 CMIP-6.2" ;
                :activity_id = "ISMIP6" ;
                :branch_method = "no parent" ;
                :branch_time_in_child = 59400. ;
                :branch_time_in_parent = 0. ;
                :contact = "Python Coder (coder@a.b.c.com)" ;
                :creation_date = "2019-01-08T23:41:05Z" ;
                :data_specs_version = "01.00.27" ;
                :experiment = "preindustrial control with interactive ice shee
t" ;
                :experiment_id = "piControl-withism" ;
                :external_variables = "areacella" ;
                :forcing_index = 1 ;
                :frequency = "mon" ;
                :further_info_url = "https://furtherinfo.es-doc.org/CMIP6.PCMD
I.PCMDI-test-1-0.piControl-withism.none.r3i1p1f1" ;
                :grid = "native atmosphere regular grid (3x4 latxlon)" ;
                :grid_label = "gn" ;
                :history = "2019-01-08T23:41:05Z ;rewrote data to be consistent
with ISMIP6 for variable tas found in table Amon.;\n",
                        "Output from archivcl_A1.nce/giccm_03_std_2xCO2_2256." ;
                :initialization_index = 1 ;
                :institution = "Program for Climate Model Diagnosis and Intercom
parison, Lawrence Livermore National Laboratory, Livermore, CA 94550, USA" ;
                :institution_id = "PCMDI" ;
                :mip_era = "CMIP6" ;
                :nominal_resolution = "10000 km" ;
                :parent_activity_id = "no parent" ;
                :parent_experiment_id = "no parent" ;
                :parent_mip_era = "no parent" ;
                :parent_source_id = "no parent" ;
                :parent_time_units = "no parent" ;
                :parent_variant_label = "no parent" ;
                :physics_index = 1 ;
                :product = "model-output" ;
                :realization_index = 3 ;
                :realm = "atmos" ;
                :references = "Model described by Koder and Tolkien (J. Geophy
s. Res., 2001, 576-591).  Also see http://www.GICC.su/giccm/doc/index.html.  Th
e ssp245 simulation is described in Dorkey et al. \'(Clim. Dyn., 2003, 323-35
7.)\'" ;
                :run_variant = "3rd realization" ;
                :source = "PCMDI-test 1.0 (1989): \n",
                        "aerosol: none\n",
                        "atmos: Earth1.0-gettingHotter (360 x 180 longitude/lati
tude; 50 levels; top level 0.1 mb)\n",
                        "atmosChem: none\n",
                        "land: Earth1.0\n",
                        "landIce: none\n",
```

```
                        "ocean: BlueMarble1.0-warming (360 x 180 longitude/latit
ude; 50 levels; top grid cell 0-10 m)\n",
                        "ocnBgchem: none\n",
                        "seaIce: Declining1.0-warming (360 x 180 longitude/latit
ude)" ;
                :source_id = "PCMDI-test-1-0" ;
                :source_type = "AOGCM ISM AER" ;
                :sub_experiment = "none" ;
                :sub_experiment_id = "none" ;
                :table_id = "Amon" ;
                :table_info = "Creation Date:(30 July 2018) MD5:fa9bc503f57fb067
bf398cab2c4ba77e" ;
                :title = "PCMDI-test-1-0 output prepared for CMIP6" ;
                :tracking_id = "hdl:21.14100/f93e4db7-d6e5-404d-983b-dbfebc93225
0" ;
                :variable_id = "tas" ;
                :variant_label = "r3i1p1f1" ;
                :license = "CMIP6 model data produced by Lawrence Livermore PCMD
I is licensed under a Creative Commons Attribution ShareAlike 4.0 International
License (https://creativecommons.org/licenses). Consult https://pcmdi.llnl.gov/C
MIP6/TermsOfUse for terms of use governing CMIP6 output, including citation requ
irements and proper acknowledgment. Further information about this data, includi
ng some limitations, can be found via the further_info_url (recorded as a globa
l attribute in this file) and at https:///pcmdi.llnl.gov/. The data producers an
d data providers make no warranty, either express or implied, including, but no
t limited to, warranties of merchantability and fitness for a particular purpos
e. All liabilities arising from the supply of the information (including any lia
bility arising in negligence) are excluded to the fullest extent permitted by la
w." ;
                :cmor_version = "3.4.0" ;
data:

 time = 15.5, 45.5 ;

 time_bnds =
  0, 31,
  31, 60 ;

 lat = 10, 20, 30 ;

 lat_bnds =
  5, 15,
  15, 25,
  25, 35 ;

 lon = 0, 90, 180, 270 ;

 lon_bnds =
  -45, 45,
  45, 135,
```

```
  135, 225,
  225, 315 ;

 height = 2 ;

 tas =
  254.0895, 258.4085, 250.5549, 258.7101,
  258.668, 258.299, 252.1237, 255.0432,
  253.7254, 251.246, 254.3168, 255.4808,
  259.7908, 252.2754, 257.1892, 253.3132,
  253.8823, 253.4698, 253.5381, 254.973,
  256.1002, 251.8168, 259.3698, 250.2994 ;
}
```

## Example 5: Treatment of Auxiliary Coordinates (northward ocean heat transport; a function of latitude, ocean basin, month)

- example5.py (page 0)

**Click to expand Python code**

```
import cmor
import numpy
import os

data = 10. * numpy.random.random_sample((2, 3, 4)) + 250.
data = numpy.array([-80, -84, -88,
                    -100, -104, -76,
                    -120, -92, -96,
                    -79, -83, -87,
                    -99, -103, -75,
                    -107, -111, -115
                    ])
data.shape = (2, 3, 3)
lat = numpy.array([10, 20, 30])
lat_bnds = numpy.array([5, 15, 25, 35])
time = numpy.array([15.5, 45])
time_bnds = numpy.array([0, 31, 60])
region = [
    "atlantic_arctic_ocean",
    "indian_pacific_ocean",
    "global_ocean"
]
ipth = opth = 'Test'
cmor.setup(inpath=ipth,
           set_verbosity=cmor.CMOR_NORMAL,
           netcdf_file_action=cmor.CMOR_REPLACE)
cmor.dataset_json("CMOR_input_example.json")
cmor.load_table("CMIP6_Omon.json")
cmorLat = cmor.axis("latitude",
                    coord_vals=lat,
                    cell_bounds=lat_bnds,
                    units="degrees_north")
cmorTime = cmor.axis("time",
                     coord_vals=time,
                     cell_bounds=time_bnds,
                     units="days since 2018")
cmorBasin = cmor.axis("basin", coord_vals=region, units="")
axes = [cmorTime, cmorBasin, cmorLat]
cmorVar = cmor.variable("htovgyre", "W", axes)
cmor.write(cmorVar, data)
filename = cmor.close(cmorVar, file_name=True)
print("Stored in:", filename)
cmor.close()
os.system("ncdump {}".format(filename))
```

**Click to expand NetCDF dump**

```
netcdf htovgyre_Omon_PCMDI-test-1-0_piControl-withism_r3i1p1f1_gn_201801-201802
{
dimensions:
        time = UNLIMITED ; // (2 currently)
        basin = 3 ;
        lat = 3 ;
        bnds = 2 ;
        strlen = 21 ;
variables:
        double time(time) ;
                time:bounds = "time_bnds" ;
                time:units = "days since 2018" ;
                time:calendar = "360_day" ;
                time:axis = "T" ;
                time:long_name = "time" ;
                time:standard_name = "time" ;
        double time_bnds(time, bnds) ;
        char sector(basin, strlen) ;
                sector:long_name = "ocean basin" ;
                sector:standard_name = "region" ;
        double lat(lat) ;
                lat:bounds = "lat_bnds" ;
                lat:units = "degrees_north" ;
                lat:axis = "Y" ;
                lat:long_name = "latitude" ;
                lat:standard_name = "latitude" ;
        double lat_bnds(lat, bnds) ;
        float htovgyre(time, basin, lat) ;
                htovgyre:standard_name = "northward_ocean_heat_transport_due_t
o_gyre" ;
                htovgyre:long_name = "Northward Ocean Heat Transport due to Gyr
e" ;
                htovgyre:comment = "From all advective mass transport processe
s, resolved and parameterized." ;
                htovgyre:units = "W" ;
                htovgyre:cell_methods = "longitude: mean time: mean" ;
                htovgyre:missing_value = 1.e+20f ;
                htovgyre:_FillValue = 1.e+20f ;
                htovgyre:history = "2019-01-08T23:45:26Z altered by CMOR: Conver
ted type from \'l\' to \'f\'." ;
                htovgyre:coordinates = "sector" ;

// global attributes:
                :Conventions = "CF-1.7 CMIP-6.2" ;
                :activity_id = "ISMIP6" ;
                :branch_method = "no parent" ;
                :branch_time_in_child = 59400. ;
                :branch_time_in_parent = 0. ;
                :contact = "Python Coder (coder@a.b.c.com)" ;
                :creation_date = "2019-01-08T23:45:26Z" ;
```

```
                :data_specs_version = "01.00.27" ;
                :experiment = "preindustrial control with interactive ice shee
t" ;
                :experiment_id = "piControl-withism" ;
                :forcing_index = 1 ;
                :frequency = "mon" ;
                :further_info_url = "https://furtherinfo.es-doc.org/CMIP6.PCMD
I.PCMDI-test-1-0.piControl-withism.none.r3i1p1f1" ;
                :grid = "native atmosphere regular grid (3x4 latxlon)" ;
                :grid_label = "gn" ;
                :history = "2019-01-08T23:45:26Z ;rewrote data to be consistent
with ISMIP6 for variable htovgyre found in table Omon.;\n",
                        "Output from archivcl_A1.nce/giccm_03_std_2xCO2_2256." ;
                :initialization_index = 1 ;
                :institution = "Program for Climate Model Diagnosis and Intercom
parison, Lawrence Livermore National Laboratory, Livermore, CA 94550, USA" ;
                :institution_id = "PCMDI" ;
                :mip_era = "CMIP6" ;
                :nominal_resolution = "10000 km" ;
                :parent_activity_id = "no parent" ;
                :parent_experiment_id = "no parent" ;
                :parent_mip_era = "no parent" ;
                :parent_source_id = "no parent" ;
                :parent_time_units = "no parent" ;
                :parent_variant_label = "no parent" ;
                :physics_index = 1 ;
                :product = "model-output" ;
                :realization_index = 3 ;
                :realm = "ocean" ;
                :references = "Model described by Koder and Tolkien (J. Geophy
s. Res., 2001, 576-591).  Also see http://www.GICC.su/giccm/doc/index.html.  Th
e ssp245 simulation is described in Dorkey et al. \'(Clim. Dyn., 2003, 323-35
7.)\'" ;
                :run_variant = "3rd realization" ;
                :source = "PCMDI-test 1.0 (1989): \n",
                        "aerosol: none\n",
                        "atmos: Earth1.0-gettingHotter (360 x 180 longitude/lati
tude; 50 levels; top level 0.1 mb)\n",
                        "atmosChem: none\n",
                        "land: Earth1.0\n",
                        "landIce: none\n",
                        "ocean: BlueMarble1.0-warming (360 x 180 longitude/latit
ude; 50 levels; top grid cell 0-10 m)\n",
                        "ocnBgchem: none\n",
                        "seaIce: Declining1.0-warming (360 x 180 longitude/latit
ude)" ;
                :source_id = "PCMDI-test-1-0" ;
                :source_type = "AOGCM ISM AER" ;
                :sub_experiment = "none" ;
                :sub_experiment_id = "none" ;
```

```
                :table_id = "Omon" ;
                :table_info = "Creation Date:(30 July 2018) MD5:fa9bc503f57fb067
bf398cab2c4ba77e" ;
                :title = "PCMDI-test-1-0 output prepared for CMIP6" ;
                :tracking_id = "hdl:21.14100/631e76b6-64a0-4f24-8c67-e3a9a03a292
0" ;
                :variable_id = "htovgyre" ;
                :variant_label = "r3i1p1f1" ;
                :license = "CMIP6 model data produced by Lawrence Livermore PCMD
I is licensed under a Creative Commons Attribution ShareAlike 4.0 International
License (https://creativecommons.org/licenses). Consult https://pcmdi.llnl.gov/C
MIP6/TermsOfUse for terms of use governing CMIP6 output, including citation requ
irements and proper acknowledgment. Further information about this data, includi
ng some limitations, can be found via the further_info_url (recorded as a globa
l attribute in this file) and at https:///pcmdi.llnl.gov/. The data producers an
d data providers make no warranty, either express or implied, including, but no
t limited to, warranties of merchantability and fitness for a particular purpos
e. All liabilities arising from the supply of the information (including any lia
bility arising in negligence) are excluded to the fullest extent permitted by la
w." ;
                :cmor_version = "3.4.0" ;
data:

 time = 15.5, 45.5 ;

 time_bnds =
  0, 31,
  31, 60 ;

 sector =
  "atlantic_arctic_ocean",
  "indian_pacific_ocean",
  "global_ocean" ;

 lat = 10, 20, 30 ;

 lat_bnds =
  5, 15,
  15, 25,
  25, 35 ;

 htovgyre =
  -80, -84, -88,
  -100, -104, -76,
  -120, -92, -96,
  -79, -83, -87,
  -99, -103, -75,
  -107, -111, -115 ;
}
```

## Example 6: Treatment of a 3-D Field on Model Levels (cloud fraction; a function of longitude, latitude, model level, month)

- example6.py (page 0)

**Click to expand Python code**

```
import cmor
import numpy
import os

data = 10. * numpy.random.random_sample((2, 3, 4)) + 250.
data = numpy.array([
    72.8, 73.2, 73.6, 74,
    71.6, 72, 72.4, 72.4,
    70.4, 70.8, 70.8, 71.2,
    67.6, 69.2, 69.6, 70,
    66, 66.4, 66.8, 67.2,
    64.8, 65.2, 65.6, 66,
    63.6, 64, 64.4, 64.4,
    60.8, 61.2, 62.8, 63.2,
    59.6, 59.6, 60, 60.4,
    58, 58.4, 58.8, 59.2,
    56.8, 57.2, 57.6, 58,
    54, 54.4, 54.8, 56.4,
    52.8, 53.2, 53.2, 53.6,
    51.6, 51.6, 52, 52.4,
    50, 50.4, 50.8, 51.2,
    72.9, 73.3, 73.7, 74.1,
    71.7, 72.1, 72.5, 72.5,
    70.5, 70.9, 70.9, 71.3,
    67.7, 69.3, 69.7, 70.1,
    66.1, 66.5, 66.9, 67.3,
    64.9, 65.3, 65.7, 66.1,
    63.7, 64.1, 64.5, 64.5,
    60.9, 61.3, 62.9, 63.3,
    59.7, 59.7, 60.1, 60.5,
    58.1, 58.5, 58.9, 59.3,
    56.9, 57.3, 57.7, 58.1,
    54.1, 54.5, 54.9, 56.5,
    52.9, 53.3, 53.3, 53.7,
    51.7, 51.7, 52.1, 52.5,
    50.1, 50.5, 50.9, 51.3])
data.shape = (2, 5, 3, 4)
lat = numpy.array([10, 20, 30])
lat_bnds = numpy.array([5, 15, 25, 35])
lon = numpy.array([0, 90, 180, 270])
lon_bnds = numpy.array([-45, 45,
                        135,
                        225,
                        315
                        ])
time = numpy.array([15.5, 45])
time_bnds = numpy.array([0, 31, 60])
lev = [0.92, 0.72, 0.5, 0.3, 0.1]
lev_bnds = [1, 0.83,
            0.61,
```

```
            0.4,
            0.2,
            0
            ]
p0 = 100000
a = [0.12, 0.22, 0.3, 0.2, 0.1]
b = [0.8, 0.5, 0.2, 0.1, 0]
ps = numpy.array([
    97000, 97400, 97800, 98200,
    98600, 99000, 99400, 99800,
    100200, 100600, 101000, 101400,
    97100, 97500, 97900, 98300,
    98700, 99100, 99500, 99900,
    100300, 100700, 101100, 101500])
ps.shape = (2, 3, 4)
a_bnds = [
    0.06, 0.18,
    0.26,
    0.25,
    0.15,
    0]
b_bnds = [
    0.94, 0.65,
    0.35,
    0.15,
    0.05,
    0]
ipth = opth = 'Test'
cmor.setup(inpath=ipth,
           set_verbosity=cmor.CMOR_NORMAL,
           netcdf_file_action=cmor.CMOR_REPLACE)
cmor.dataset_json("CMOR_input_example.json")
cmor.load_table("CMIP6_Amon.json")
cmorLat = cmor.axis("latitude",
                    coord_vals=lat,
                    cell_bounds=lat_bnds,
                    units="degrees_north")
cmorLon = cmor.axis("longitude",
                    coord_vals=lon,
                    cell_bounds=lon_bnds,
                    units="degrees_east")
cmorTime = cmor.axis("time",
                     coord_vals=time,
                     cell_bounds=time_bnds,
                     units="days since 2018")
cmorLev = cmor.axis("standard_hybrid_sigma",
                    units='1',
                    coord_vals=lev,
                    cell_bounds=lev_bnds)
axes = [cmorTime, cmorLev, cmorLat, cmorLon]
```

```
ierr = cmor.zfactor(zaxis_id=cmorLev,
                    zfactor_name='a',
                    axis_ids=[cmorLev, ],
                    zfactor_values=a,
                    zfactor_bounds=a_bnds)
ierr = cmor.zfactor(zaxis_id=cmorLev,
                    zfactor_name='b',
                    axis_ids=[cmorLev, ],
                    zfactor_values=b,
                    zfactor_bounds=b_bnds)
ierr = cmor.zfactor(zaxis_id=cmorLev,
                    zfactor_name='p0',
                    units='Pa',
                    zfactor_values=p0)
ips = cmor.zfactor(zaxis_id=cmorLev,
                   zfactor_name='ps',
                   axis_ids=[cmorTime, cmorLat, cmorLon],
                   units='Pa')
cmorVar = cmor.variable("cl", "%", axes)
cmor.write(cmorVar, data)
cmor.write(ips, ps, store_with=cmorVar)
filename = cmor.close(cmorVar, file_name=True)
print("Stored in:", filename)
cmor.close()
os.system("ncdump {}".format(filename))
```

**Click to expand NetCDF dump**

```
netcdf cl_Amon_PCMDI-test-1-0_piControl-withism_r3i1p1f1_gn_201801-201802 {
dimensions:
        time = UNLIMITED ; // (2 currently)
        lev = 5 ;
        lat = 3 ;
        lon = 4 ;
        bnds = 2 ;
variables:
        double time(time) ;
                time:bounds = "time_bnds" ;
                time:units = "days since 2018" ;
                time:calendar = "360_day" ;
                time:axis = "T" ;
                time:long_name = "time" ;
                time:standard_name = "time" ;
        double time_bnds(time, bnds) ;
        double lev(lev) ;
                lev:bounds = "lev_bnds" ;
                lev:units = "1" ;
                lev:axis = "Z" ;
                lev:positive = "down" ;
                lev:long_name = "hybrid sigma pressure coordinate" ;
                lev:standard_name = "atmosphere_hybrid_sigma_pressure_coordinat
e" ;
                lev:formula = "p = a*p0 + b*ps" ;
                lev:formula_terms = "p0: p0 a: a b: b ps: ps" ;
        double lev_bnds(lev, bnds) ;
                lev_bnds:formula = "p = a*p0 + b*ps" ;
                lev_bnds:standard_name = "atmosphere_hybrid_sigma_pressure_coord
inate" ;
                lev_bnds:units = "1" ;
                lev_bnds:formula_terms = "p0: p0 a: a_bnds b: b_bnds ps: ps" ;
        double p0 ;
                p0:long_name = "vertical coordinate formula term: reference pres
sure" ;
                p0:units = "Pa" ;
        double a(lev) ;
                a:long_name = "vertical coordinate formula term: a(k)" ;
        double b(lev) ;
                b:long_name = "vertical coordinate formula term: b(k)" ;
        float ps(time, lat, lon) ;
                ps:long_name = "Surface Air Pressure" ;
                ps:units = "Pa" ;
        double a_bnds(lev, bnds) ;
                a_bnds:long_name = "vertical coordinate formula term: a(k+1/2)"
;
        double b_bnds(lev, bnds) ;
                b_bnds:long_name = "vertical coordinate formula term: b(k+1/2)"
;
        double lat(lat) ;
```

```
                lat:bounds = "lat_bnds" ;
                lat:units = "degrees_north" ;
                lat:axis = "Y" ;
                lat:long_name = "latitude" ;
                lat:standard_name = "latitude" ;
        double lat_bnds(lat, bnds) ;
        double lon(lon) ;
                lon:bounds = "lon_bnds" ;
                lon:units = "degrees_east" ;
                lon:axis = "X" ;
                lon:long_name = "Longitude" ;
                lon:standard_name = "longitude" ;
        double lon_bnds(lon, bnds) ;
        float cl(time, lev, lat, lon) ;
                cl:standard_name = "cloud_area_fraction_in_atmosphere_layer" ;
                cl:long_name = "Cloud Area Fraction" ;
                cl:comment = "Percentage cloud cover, including both large-scal
e and convective cloud." ;
                cl:units = "%" ;
                cl:cell_methods = "area: time: mean" ;
                cl:cell_measures = "area: areacella" ;
                cl:missing_value = 1.e+20f ;
                cl:_FillValue = 1.e+20f ;
                cl:history = "2019-01-08T23:49:05Z altered by CMOR: Converted ty
pe from \'d\' to \'f\'." ;

// global attributes:
                :Conventions = "CF-1.7 CMIP-6.2" ;
                :activity_id = "ISMIP6" ;
                :branch_method = "no parent" ;
                :branch_time_in_child = 59400. ;
                :branch_time_in_parent = 0. ;
                :contact = "Python Coder (coder@a.b.c.com)" ;
                :creation_date = "2019-01-08T23:49:05Z" ;
                :data_specs_version = "01.00.27" ;
                :experiment = "preindustrial control with interactive ice shee
t" ;
                :experiment_id = "piControl-withism" ;
                :external_variables = "areacella" ;
                :forcing_index = 1 ;
                :frequency = "mon" ;
                :further_info_url = "https://furtherinfo.es-doc.org/CMIP6.PCMD
I.PCMDI-test-1-0.piControl-withism.none.r3i1p1f1" ;
                :grid = "native atmosphere regular grid (3x4 latxlon)" ;
                :grid_label = "gn" ;
                :history = "2019-01-08T23:49:05Z ;rewrote data to be consistent
with ISMIP6 for variable cl found in table Amon.;\n",
                        "Output from archivcl_A1.nce/giccm_03_std_2xCO2_2256." ;
                :initialization_index = 1 ;
                :institution = "Program for Climate Model Diagnosis and Intercom
```

```
parison, Lawrence Livermore National Laboratory, Livermore, CA 94550, USA" ;
                :institution_id = "PCMDI" ;
                :mip_era = "CMIP6" ;
                :nominal_resolution = "10000 km" ;
                :parent_activity_id = "no parent" ;
                :parent_experiment_id = "no parent" ;
                :parent_mip_era = "no parent" ;
                :parent_source_id = "no parent" ;
                :parent_time_units = "no parent" ;
                :parent_variant_label = "no parent" ;
                :physics_index = 1 ;
                :product = "model-output" ;
                :realization_index = 3 ;
                :realm = "atmos" ;
                :references = "Model described by Koder and Tolkien (J. Geophy
s. Res., 2001, 576-591).  Also see http://www.GICC.su/giccm/doc/index.html.  Th
e ssp245 simulation is described in Dorkey et al. \'(Clim. Dyn., 2003, 323-35
7.)\'" ;
                :run_variant = "3rd realization" ;
                :source = "PCMDI-test 1.0 (1989): \n",
                        "aerosol: none\n",
                        "atmos: Earth1.0-gettingHotter (360 x 180 longitude/lati
tude; 50 levels; top level 0.1 mb)\n",
                        "atmosChem: none\n",
                        "land: Earth1.0\n",
                        "landIce: none\n",
                        "ocean: BlueMarble1.0-warming (360 x 180 longitude/latit
ude; 50 levels; top grid cell 0-10 m)\n",
                        "ocnBgchem: none\n",
                        "seaIce: Declining1.0-warming (360 x 180 longitude/latit
ude)" ;
                :source_id = "PCMDI-test-1-0" ;
                :source_type = "AOGCM ISM AER" ;
                :sub_experiment = "none" ;
                :sub_experiment_id = "none" ;
                :table_id = "Amon" ;
                :table_info = "Creation Date:(30 July 2018) MD5:fa9bc503f57fb067
bf398cab2c4ba77e" ;
                :title = "PCMDI-test-1-0 output prepared for CMIP6" ;
                :variable_id = "cl" ;
                :variant_label = "r3i1p1f1" ;
                :license = "CMIP6 model data produced by Lawrence Livermore PCMD
I is licensed under a Creative Commons Attribution ShareAlike 4.0 International
License (https://creativecommons.org/licenses). Consult https://pcmdi.llnl.gov/C
MIP6/TermsOfUse for terms of use governing CMIP6 output, including citation requ
irements and proper acknowledgment. Further information about this data, includi
ng some limitations, can be found via the further_info_url (recorded as a globa
l attribute in this file) and at https:///pcmdi.llnl.gov/. The data producers an
d data providers make no warranty, either express or implied, including, but no
t limited to, warranties of merchantability and fitness for a particular purpos
```

```
e. All liabilities arising from the supply of the information (including any lia
bility arising in negligence) are excluded to the fullest extent permitted by la
w." ;
                :cmor_version = "3.4.0" ;
                :tracking_id = "hdl:21.14100/68486bc9-5ee7-4a03-ba74-ec9cf9c86e3
c" ;
data:

 time = 15.5, 45.5 ;

 time_bnds =
  0, 31,
  31, 60 ;

 lev = 0.92, 0.72, 0.5, 0.3, 0.1 ;

 lev_bnds =
  1, 0.83,
  0.83, 0.61,
  0.61, 0.4,
  0.4, 0.2,
  0.2, 0 ;

 p0 = 100000 ;

 a = 0.12, 0.22, 0.3, 0.2, 0.1 ;

 b = 0.8, 0.5, 0.2, 0.1, 0 ;

 ps =
  97000, 97400, 97800, 98200,
  98600, 99000, 99400, 99800,
  100200, 100600, 101000, 101400,
  97100, 97500, 97900, 98300,
  98700, 99100, 99500, 99900,
  100300, 100700, 101100, 101500 ;

 a_bnds =
  0.06, 0.18,
  0.18, 0.26,
  0.26, 0.25,
  0.25, 0.15,
  0.15, 0 ;

 b_bnds =
  0.94, 0.65,
  0.65, 0.35,
  0.35, 0.15,
  0.15, 0.05,
  0.05, 0 ;
```

```
 lat = 10, 20, 30 ;

 lat_bnds =
  5, 15,
  15, 25,
  25, 35 ;

 lon = 0, 90, 180, 270 ;

 lon_bnds =
  -45, 45,
  45, 135,
  135, 225,
  225, 315 ;

 cl =
  72.8, 73.2, 73.6, 74,
  71.6, 72, 72.4, 72.4,
  70.4, 70.8, 70.8, 71.2,
  67.6, 69.2, 69.6, 70,
  66, 66.4, 66.8, 67.2,
  64.8, 65.2, 65.6, 66,
  63.6, 64, 64.4, 64.4,
  60.8, 61.2, 62.8, 63.2,
  59.6, 59.6, 60, 60.4,
  58, 58.4, 58.8, 59.2,
  56.8, 57.2, 57.6, 58,
  54, 54.4, 54.8, 56.4,
  52.8, 53.2, 53.2, 53.6,
  51.6, 51.6, 52, 52.4,
  50, 50.4, 50.8, 51.2,
  72.9, 73.3, 73.7, 74.1,
  71.7, 72.1, 72.5, 72.5,
  70.5, 70.9, 70.9, 71.3,
  67.7, 69.3, 69.7, 70.1,
  66.1, 66.5, 66.9, 67.3,
  64.9, 65.3, 65.7, 66.1,
  63.7, 64.1, 64.5, 64.5,
  60.9, 61.3, 62.9, 63.3,
  59.7, 59.7, 60.1, 60.5,
  58.1, 58.5, 58.9, 59.3,
  56.9, 57.3, 57.7, 58.1,
  54.1, 54.5, 54.9, 56.5,
  52.9, 53.3, 53.3, 53.7,
  51.7, 51.7, 52.1, 52.5,
  50.1, 50.5, 50.9, 51.3 ;
}
```

## Example 7: Treatment of Grid Coordinates

- example7.py (page 0)

**Click to expand Python code**

```
import cmor
import numpy
import os


var_data = numpy.random.random((3, 4, 2)) + 34.

x = numpy.arange(4, dtype=float)
x_bnds = numpy.array([[x_ - 0.5, x_ + 0.5] for x_ in x])
y = numpy.arange(3, dtype=float)
y_bnds = numpy.array([[y_ - 0.5, y_ + 0.5] for y_ in y])

lat = numpy.array([10, 20, 30]).reshape((3, 1))
lat = numpy.tile(lat, (1, 4))
lon = numpy.array([0, 90, 180, 270])
lon = numpy.tile(lon, (3, 1))
lon_bnds = numpy.zeros((3, 4, 4))
lat_bnds = numpy.zeros((3, 4, 4))

for j in range(3):
    for i in range(4):
        lon_bnds[j, i, 0] = (lon[j, i] - 45) % 360
        lon_bnds[j, i, 1] = lon[j, i]
        lon_bnds[j, i, 2] = (lon[j, i] + 45) % 360
        lon_bnds[j, i, 3] = lon[j, i]
        lat_bnds[j, i, 0] = lat[j, i]
        lat_bnds[j, i, 1] = lat[j, i] - 5
        lat_bnds[j, i, 2] = lat[j, i]
        lat_bnds[j, i, 3] = lat[j, i] + 5

time = numpy.array([0, 1])
time_bnds = numpy.array([0, 1, 2])

ipth = opth = 'Test'
cmor.setup(inpath=ipth,
           set_verbosity=cmor.CMOR_NORMAL,
           netcdf_file_action=cmor.CMOR_REPLACE,
           exit_control=cmor.CMOR_EXIT_ON_MAJOR)
cmor.dataset_json('CMOR_input_example.json')

# First, load the grids table to set up x and y axes and the lat-long grid
grid_table_id = cmor.load_table('CMIP6_grids.json')
cmor.set_table(grid_table_id)

y_axis_id = cmor.axis(table_entry='y_deg',
                      units='degrees',
                      coord_vals=y,
                      cell_bounds=y_bnds)
x_axis_id = cmor.axis(table_entry='x_deg',
                      units='degrees',
```

```
                      coord_vals=x,
                      cell_bounds=x_bnds)

grid_id = cmor.grid(axis_ids=[y_axis_id, x_axis_id],
                    latitude=lat,
                    longitude=lon,
                    latitude_vertices=lat_bnds,
                    longitude_vertices=lon_bnds)

# Now, load the Omon table to set up the time axis and variable
omon_table_id = cmor.load_table('CMIP6_Omon.json')
cmor.set_table(omon_table_id)

time_axis_id = cmor.axis(table_entry='time',
                         units='months since 1980',
                         coord_vals=time,
                         cell_bounds=time_bnds)

var_id = cmor.variable(table_entry='sos',
                       units='0.001',
                       axis_ids=[grid_id, time_axis_id])

cmor.write(var_id, var_data, 2)

filename = cmor.close(var_id, file_name=True)
print("Stored in:", filename)
cmor.close()
os.system("ncdump {}".format(filename))
```

**Click to expand NetCDF dump**

```
netcdf sos_Omon_PCMDI-test-1-0_piControl-withism_r3i1p1f1_gn_198001-198002 {
dimensions:
        time = UNLIMITED ; // (2 currently)
        y = 3 ;
        x = 4 ;
        bnds = 2 ;
        vertices = 4 ;
variables:
        double time(time) ;
                time:bounds = "time_bnds" ;
                time:units = "days since 1980" ;
                time:calendar = "360_day" ;
                time:axis = "T" ;
                time:long_name = "time" ;
                time:standard_name = "time" ;
        double time_bnds(time, bnds) ;
        double y(y) ;
                y:bounds = "y_bnds" ;
                y:units = "degrees" ;
                y:axis = "Y" ;
                y:long_name = "y coordinate of projection" ;
                y:standard_name = "projection_y_coordinate" ;
        double y_bnds(y, bnds) ;
        double x(x) ;
                x:bounds = "x_bnds" ;
                x:units = "degrees" ;
                x:axis = "X" ;
                x:long_name = "x coordinate of projection" ;
                x:standard_name = "projection_x_coordinate" ;
        double x_bnds(x, bnds) ;
        double latitude(y, x) ;
                latitude:standard_name = "latitude" ;
                latitude:long_name = "latitude" ;
                latitude:units = "degrees_north" ;
                latitude:missing_value = 1.e+20 ;
                latitude:_FillValue = 1.e+20 ;
                latitude:bounds = "vertices_latitude" ;
        double longitude(y, x) ;
                longitude:standard_name = "longitude" ;
                longitude:long_name = "longitude" ;
                longitude:units = "degrees_east" ;
                longitude:missing_value = 1.e+20 ;
                longitude:_FillValue = 1.e+20 ;
                longitude:bounds = "vertices_longitude" ;
        double vertices_latitude(y, x, vertices) ;
                vertices_latitude:units = "degrees_north" ;
                vertices_latitude:missing_value = 1.e+20 ;
                vertices_latitude:_FillValue = 1.e+20 ;
        double vertices_longitude(y, x, vertices) ;
                vertices_longitude:units = "degrees_east" ;
```

```
                vertices_longitude:missing_value = 1.e+20 ;
                vertices_longitude:_FillValue = 1.e+20 ;
        float sos(time, y, x) ;
                sos:standard_name = "sea_surface_salinity" ;
                sos:long_name = "Sea Surface Salinity" ;
                sos:comment = "Sea water salinity is the salt content of sea wat
er, often on the Practical Salinity Scale of 1978. However, the unqualified ter
m \'salinity\' is generic and does not necessarily imply any particular method o
f calculation. The units of salinity are dimensionless and the units attribute s
hould normally be given as 1e-3 or 0.001 i.e. parts per thousand." ;
                sos:units = "0.001" ;
                sos:cell_methods = "area: mean where sea time: mean" ;
                sos:cell_measures = "area: areacello" ;
                sos:history = "2025-01-13T20:25:29Z altered by CMOR: Reordered d
imensions, original order: y x time. 2025-01-13T20:25:29Z altered by CMOR: Conve
rted type from \'d\' to \'f\'." ;
                sos:missing_value = 1.e+20f ;
                sos:_FillValue = 1.e+20f ;
                sos:coordinates = "latitude longitude" ;

// global attributes:
                :Conventions = "CF-1.7 CMIP-6.2" ;
                :activity_id = "ISMIP6" ;
                :branch_method = "no parent" ;
                :branch_time_in_child = 59400. ;
                :branch_time_in_parent = 0. ;
                :contact = "Python Coder (coder@a.b.c.com)" ;
                :creation_date = "2025-01-13T20:25:29Z" ;
                :data_specs_version = "01.00.33" ;
                :experiment = "preindustrial control with interactive ice shee
t" ;
                :experiment_id = "piControl-withism" ;
                :external_variables = "areacello" ;
                :forcing_index = 1 ;
                :frequency = "mon" ;
                :further_info_url = "https://furtherinfo.es-doc.org/CMIP6.PCMD
I.PCMDI-test-1-0.piControl-withism.none.r3i1p1f1" ;
                :grid = "native atmosphere regular grid (3x4 latxlon)" ;
                :grid_label = "gn" ;
                :history = "2025-01-13T20:25:29Z ;rewrote data to be consistent
with ISMIP6 for variable sos found in table Omon.;\n",
                        "Output from archivcl_A1.nce/giccm_03_std_2xCO2_2256." ;
                :initialization_index = 1 ;
                :institution = "Program for Climate Model Diagnosis and Intercom
parison, Lawrence Livermore National Laboratory, Livermore, CA 94550, USA" ;
                :institution_id = "PCMDI" ;
                :mip_era = "CMIP6" ;
                :nominal_resolution = "10000 km" ;
                :parent_activity_id = "no parent" ;
                :parent_experiment_id = "no parent" ;
```

```
                :parent_mip_era = "no parent" ;
                :parent_source_id = "no parent" ;
                :parent_time_units = "no parent" ;
                :parent_variant_label = "no parent" ;
                :physics_index = 1 ;
                :product = "model-output" ;
                :realization_index = 3 ;
                :realm = "ocean" ;
                :references = "Model described by Koder and Tolkien (J. Geophy
s. Res., 2001, 576-591).  Also see http://www.GICC.su/giccm/doc/index.html.  Th
e ssp245 simulation is described in Dorkey et al. \'(Clim. Dyn., 2003, 323-35
7.)\'" ;
                :run_variant = "3rd realization" ;
                :source = "PCMDI-test 1.0 (1989): \n",
                        "aerosol: none\n",
                        "atmos: Earth1.0-gettingHotter (360 x 180 longitude/lati
tude; 50 levels; top level 0.1 mb)\n",
                        "atmosChem: none\n",
                        "land: Earth1.0\n",
                        "landIce: none\n",
                        "ocean: BlueMarble1.0-warming (360 x 180 longitude/latit
ude; 50 levels; top grid cell 0-10 m)\n",
                        "ocnBgchem: none\n",
                        "seaIce: Declining1.0-warming (360 x 180 longitude/latit
ude)" ;
                :source_id = "PCMDI-test-1-0" ;
                :source_type = "AOGCM ISM AER" ;
                :sub_experiment = "none" ;
                :sub_experiment_id = "none" ;
                :table_id = "Omon" ;
                :table_info = "Creation Date:(18 November 2020) MD5:1a7c21fe7a3e
c7429780675800b70460" ;
                :title = "PCMDI-test-1-0 output prepared for CMIP6" ;
                :tracking_id = "hdl:21.14100/d2dac814-dcd1-4da6-a6f9-e35ebb1d1bc
b" ;
                :variable_id = "sos" ;
                :variant_label = "r3i1p1f1" ;
                :license = "CMIP6 model data produced by Lawrence Livermore PCMD
I is licensed under a Creative Commons Attribution 4.0 International License (ht
tps://creativecommons.org/licenses/by/4.0/). Consult https://pcmdi.llnl.gov/CMIP
6/TermsOfUse for terms of use governing CMIP6 output, including citation require
ments and proper acknowledgment. Further information about this data, including
some limitations, can be found via the further_info_url (recorded as a global at
tribute in this file) and at https:///pcmdi.llnl.gov/. The data producers and da
ta providers make no warranty, either express or implied, including, but not lim
ited to, warranties of merchantability and fitness for a particular purpose. Al
l liabilities arising from the supply of the information (including any liabilit
y arising in negligence) are excluded to the fullest extent permitted by law." ;
                :cmor_version = "3.9.0" ;
data:
```

```
time = 15, 45 ;

time_bnds =
 0, 30,
 30, 60 ;

y = 0, 1, 2 ;

y_bnds =
 -0.5, 0.5,
 0.5, 1.5,
 1.5, 2.5 ;

x = 0, 1, 2, 3 ;

x_bnds =
 -0.5, 0.5,
 0.5, 1.5,
 1.5, 2.5,
 2.5, 3.5 ;

latitude =
 10, 10, 10, 10,
 20, 20, 20, 20,
 30, 30, 30, 30 ;

longitude =
 0, 90, 180, 270,
 0, 90, 180, 270,
 0, 90, 180, 270 ;

vertices_latitude =
 10, 5, 10, 15,
 10, 5, 10, 15,
 10, 5, 10, 15,
 10, 5, 10, 15,
 20, 15, 20, 25,
 20, 15, 20, 25,
 20, 15, 20, 25,
 20, 15, 20, 25,
 30, 25, 30, 35,
 30, 25, 30, 35,
 30, 25, 30, 35,
 30, 25, 30, 35 ;

vertices_longitude =
 315, 0, 45, 0,
 45, 90, 135, 90,
 135, 180, 225, 180,
```

```
  225, 270, 315, 270,
  315, 0, 45, 0,
  45, 90, 135, 90,
  135, 180, 225, 180,
  225, 270, 315, 270,
  315, 0, 45, 0,
  45, 90, 135, 90,
  135, 180, 225, 180,
  225, 270, 315, 270 ;

 sos =
  34.8876, 34.84557, 34.84164, 34.83435,
  34.9416, 34.08398, 34.33438, 34.99736,
  34.23587, 34.22417, 34.60325, 34.67004,
  34.32861, 34.72874, 34.52488, 34.19158,
  34.48695, 34.07372, 34.91237, 34.68412,
  34.53342, 34.16178, 34.93912, 34.50451 ;
}
```

# Fortran Example

## CMOR user input

- CMOR_input_example.json
  (https://github.com/PCMDI/cmor/blob/main/Test/CMOR_input_example.json)

**Click to expand json file**

```
{
  "#note":           "explanation of what source_type is goes here",
  "source_type":            "AOGCM ISM AER",

  "#note":                  "CMIP6 valid experiment_ids are found in CMIP6_CV.js
on",
  "experiment_id":          "piControl-withism",
  "activity_id":            "ISMIP6",
  "sub_experiment_id":      "none",

  "realization_index":      "3",
  "initialization_index":   "1",
  "physics_index":          "1",
  "forcing_index":          "1",

  "#note":                  "Text stored in attribute variant_info (recommende
d, not required description of run variant)",
  "run_variant":            "3rd realization",

  "parent_experiment_id":   "historical",
  "parent_activity_id":     "CMIP",
  "parent_source_id":       "PCMDI-test-1-0",
  "parent_variant_label":   "r3i1p1f1",

  "parent_time_units":      "days since 1850-01-01",
  "branch_method":          "standard",
  "branch_time_in_child":   59400.0,
  "branch_time_in_parent":  59400.0,

  "#note":                  "institution_id must be registered at https://githu
b.com/WCRP-CMIP/CMIP6_CVs/issues/new ",
  "institution_id":         "PCMDI",

  "#note":                  "source_id (model name) must be registered at http
s://github.com/WCRP-CMIP/CMIP6_CVs/issues/new ",
  "source_id":              "PCMDI-test-1-0",

  "calendar":               "360_day",

  "grid":                   "native atmosphere regular grid (3x4 latxlon)",
  "grid_label":             "gn",
  "nominal_resolution":     "10000 km",

  "license":                 "CMIP6 model data produced by Lawrence Livermore PC
MDI is licensed under a Creative Commons Attribution ShareAlike 4.0 Internationa
l License (https://creativecommons.org/licenses). Consult https://pcmdi.llnl.go
v/CMIP6/TermsOfUse for terms of use governing CMIP6 output, including citation r
equirements and proper acknowledgment. Further information about this data, incl
uding some limitations, can be found via the further_info_url (recorded as a glo
bal attribute in this file) and at https:///pcmdi.llnl.gov/. The data producers
```

```
and data providers make no warranty, either express or implied, including, but n
ot limited to, warranties of merchantability and fitness for a particular purpos
e. All liabilities arising from the supply of the information (including any lia
bility arising in negligence) are excluded to the fullest extent permitted by la
w.",

  "#output":                "Root directory for output (can be either a relativ
e or full path)",
  "outpath":                "CMIP6",

  "#note":                  " **** The following descriptors are optional and ma
y be set to an empty string ",

  "contact ":               "Python Coder (coder@a.b.c.com)",
  "history":                "Output from archivcl_A1.nce/giccm_03_std_2xCO2_225
6.",
  "comment":                "",
  "references":             "Model described by Koder and Tolkien (J. Geophys. R
es., 2001, 576-591).  Also see http://www.GICC.su/giccm/doc/index.html.  The ssp
245 simulation is described in Dorkey et al. '(Clim. Dyn., 2003, 323-357.)'",

  "#note":                  " **** The following will be obtained from the CV an
d do not need to be defined here",

  "sub_experiment":         "none",
  "institution":            "",
  "source":                 "PCMDI-test 1.0 (1989)",

  "#note":                  " **** The following are set correctly for CMIP6 an
d should not normally need editing",

  "_controlled_vocabulary_file": "CMIP6_CV.json",
  "_AXIS_ENTRY_FILE":         "CMIP6_coordinate.json",
  "_FORMULA_VAR_FILE":        "CMIP6_formula_terms.json",
  "_cmip6_option":           "CMIP6",

  "mip_era":                "CMIP6",
  "parent_mip_era":         "CMIP6",

  "tracking_prefix":        "hdl:21.14100",
  "_history_template":       "%s ;rewrote data to be consistent with <activity_i
d> for variable <variable_id> found in table <table_id>.",

  "#output_path_template":   "Template for output path directory using tables ke
ys or global attributes, these should follow the relevant data reference synta
x",
  "output_path_template":    "<mip_era><activity_id><institution_id><source_i
d><experiment_id><_member_id><table><variable_id><grid_label><version>",
  "output_file_template":    "<variable_id><table><source_id><experiment_id><_me
mber_id><grid_label>",
```

```
}
```

### Fortran source code

- ipcc_test_code.f90 (https://github.com/PCMDI/cmor/blob/main/Test/ipcc_test_code.f90)
- reader_2D_3D.f90 (https://github.com/PCMDI/cmor/blob/main/Test/reader_2D_3D.f90)

**Click to expand Fortran code**

```
!!$pgf90 -I/work/NetCDF/5.1/include -L/work/NetCDF/5.1/lib -l netcdf -L. -l cmo
r Test/test_dimensionless.f90 -IModules -o cmor_test
!!$pgf90 -g -I/pcmdi/charles_work/NetCDF/include -L/pcmdi/charles_work/NetCDF/li
b -lnetcdf -module Modules -IModules -L. -lcmor -I/pcmdi/charles_work/Unidata/in
clude -L/pcmdi/charles_work/Unidata/lib -ludunits Test/test_dimensionless.f90
-o cmor_test

MODULE local_subs

USE cmor_users_functions
PRIVATE
PUBLIC read_coords, read_time, read_3d_input_files, read_2d_input_files
CONTAINS

SUBROUTINE read_coords(alats, alons, plevs, bnds_lat, bnds_lon)

  IMPLICIT NONE

  DOUBLE PRECISION, INTENT(OUT), DIMENSION(:) :: alats
  DOUBLE PRECISION, INTENT(OUT), DIMENSION(:) :: alons
  DOUBLE PRECISION, INTENT(OUT), DIMENSION(:) :: plevs
  DOUBLE PRECISION, INTENT(OUT), DIMENSION(:,:) :: bnds_lat
  DOUBLE PRECISION, INTENT(OUT), DIMENSION(:,:) :: bnds_lon

  INTEGER :: i

  DO i = 1, SIZE(alons)
      alons(i) = (i-1)*360./SIZE(alons)
      bnds_lon(1,i) = (i - 1.5)*360./SIZE(alons)
      bnds_lon(2,i) = (i - 0.5)*360./SIZE(alons)
  END DO

  DO i = 1, SIZE(alats)
      alats(i) = (size(alats)+1-i)*10
      bnds_lat(1,i) = (size(alats)+1-i)*10 + 5.
      bnds_lat(2,i) = (size(alats)+1-i)*10 - 5.
  END DO

  DO i = 1, SIZE(plevs)
      plevs(i) = i*1.0e4
  END DO
    plevs = (/100000., 92500., 85000., 70000.,&
      60000., 50000., 40000., 30000., 25000., 20000.,&
      15000., 10000., 7000., 5000., 3000., 2000., 1000., 500., 100./)

  RETURN
END SUBROUTINE read_coords

SUBROUTINE read_time(it, time, time_bnds)
```

```
  IMPLICIT NONE

  INTEGER, INTENT(IN) :: it
  DOUBLE PRECISION, INTENT(OUT) :: time
  DOUBLE PRECISION, INTENT(OUT), DIMENSION(2,1) :: time_bnds

  time = (it-0.5)*30.
  time_bnds(1,1) = (it-1)*30.
  time_bnds(2,1) = it*30.

  RETURN
END SUBROUTINE read_time

INCLUDE "reader_2D_3D.f90"

END MODULE local_subs


PROGRAM ipcc_test_code
!
!   Purpose:   To serve as a generic example of an application that
!       uses the "Climate Model Output Rewriter" (CMOR)

!    CMOR writes CF-compliant netCDF files.
!    Its use is strongly encouraged by the IPCC and is intended for use
!       by those participating in many community-coordinated standard
!       climate model experiments (e.g., AMIP, CMIP, CFMIP, PMIP, APE,
!       etc.)
!
!   Background information for this sample code:
!
!      Atmospheric standard output requested by IPCC are listed in
!   tables available on the web.  Monthly mean output is found in
!   tables A1a and A1c.  This sample code processes only two 3-d
!   variables listed in table A1c ("monthly mean atmosphere 3-D data"
!   and only four 2-d variables listed in table A1a ("monthly mean
!   atmosphere + land surface 2-D (latitude, longitude) data").  The
!   extension to many more fields is trivial.
!
!      For this example, the user must fill in the sections of code that
!   extract the 3-d and 2-d fields from his monthly mean "history"
!   files (which usually contain many variables but only a single time
!   slice).  The CMOR code will write each field in a separate file, but
!   many monthly mean time-samples will be stored together.  These
!   constraints partially determine the structure of the code.
!
!
!   Record of revisions:

!       Date        Programmer(s)           Description of change
```

```
!       ====        ==========              ====================
!      10/22/03     Rusty Koder              Original code
!       1/28/04     Les R. Koder             Revised to be consistent
!                                            with evolving code design

! include module that contains the user-accessible cmor functions.
USE cmor_users_functions
USE local_subs

IMPLICIT NONE

!   dimension parameters:
! ---------------------------------
INTEGER, PARAMETER :: ntimes = 2    ! number of time samples to process
INTEGER, PARAMETER :: lon = 4       ! number of longitude grid cells
INTEGER, PARAMETER :: lat = 3       ! number of latitude grid cells
INTEGER, PARAMETER :: lev = 5       ! number of standard pressure levels
INTEGER, PARAMETER :: lev2 = 19       ! number of standard pressure levels
INTEGER, PARAMETER :: n2d = 4       ! number of IPCC Table A1a fields to be
                                    !     output.
INTEGER, PARAMETER :: n3d = 3       ! number of IPCC Table A1c fields to
                                    !     be output.

!   Tables associating the user's variables with IPCC standard output
!   variables.  The user may choose to make this association in a
!   different way (e.g., by defining values of pointers that allow him
!   to directly retrieve data from a data record containing many
!   different variables), but in some way the user will need to map his
!   model output onto the Tables specifying the MIP standard output.

! ---------------------------------

                                ! My variable names for IPCC Table A1c fields
CHARACTER (LEN=5), DIMENSION(n3d) :: &
                                  varin3d=(/'CLOUD', 'U    ', 'T    '/)

                                ! Units appropriate to my data
CHARACTER (LEN=5), DIMENSION(n3d) :: &
                                  units3d=(/ '%    ', 'm s-1',   'K    '  /)

                     ! Corresponding IPCC Table A1c entry (variable name)
CHARACTER (LEN=2), DIMENSION(n3d) :: entry3d = (/ 'cl', 'ua', 'ta' /)

                                ! My variable names for IPCC Table A1a fields
CHARACTER (LEN=8), DIMENSION(n2d) :: &
                varin2d=(/ 'LATENT  ', 'TSURF   ', 'SOIL_WET', 'PSURF   ' /)

                                ! Units appropriate to my data
  CHARACTER (LEN=6), DIMENSION(n2d) :: &
                        units2d=(/ 'W m-2 ', 'K     ', 'kg m-2', 'Pa    ' /)
```

```
  CHARACTER (LEN=4), DIMENSION(n2d) :: &
                   positive2d= (/  'down',  '    ', '    ', '    '  /)

                   ! Corresponding IPCC Table A1a entry (variable name)
CHARACTER (LEN=5), DIMENSION(n2d) :: &
                     entry2d = (/ 'hfls ', 'tas  ', 'mrsos', 'ps   ' /)

!  uninitialized variables used in communicating with CMOR:
!  ---------------------------------------------------------

INTEGER :: error_flag
INTEGER :: znondim_id, zfactor_id
INTEGER, DIMENSION(n2d) :: var2d_ids
INTEGER, DIMENSION(n3d) :: var3d_ids
REAL, DIMENSION(lon,lat) :: data2d
REAL, DIMENSION(lon,lat,lev2) :: data3d
DOUBLE PRECISION, DIMENSION(lat) :: alats
DOUBLE PRECISION, DIMENSION(lon) :: alons
DOUBLE PRECISION, DIMENSION(lev2) :: plevs
DOUBLE PRECISION, DIMENSION(1) :: time
DOUBLE PRECISION, DIMENSION(2,1):: bnds_time
DOUBLE PRECISION, DIMENSION(2,lat) :: bnds_lat
DOUBLE PRECISION, DIMENSION(2,lon) :: bnds_lon
DOUBLE PRECISION, DIMENSION(lev) :: zlevs
DOUBLE PRECISION, DIMENSION(lev+1) :: zlev_bnds
REAL, DIMENSION(lev) :: a_coeff
REAL, DIMENSION(lev) :: b_coeff
REAL :: p0
REAL, DIMENSION(lev+1) :: a_coeff_bnds
REAL, DIMENSION(lev+1) :: b_coeff_bnds
INTEGER :: ilon, ilat, ipres, ilev, itim, itim2, ilon2,ilat2
DOUBLE PRECISION bt

character(256)::  outpath,mycal

!  Other variables:
!  --------------------

INTEGER :: it, m
bt=0.
! ================================
!  Execution begins here:
! ================================

! Read coordinate information from model into arrays that will be passed
!   to CMOR.
! Read latitude, longitude, and pressure coordinate values into
!   alats, alons, and plevs, respectively.  Also generate latitude and
!   longitude bounds, and store in bnds_lat and bnds_lon, respectively.
```

```
!   Note that all variable names in this code can be freely chosen by
!   the user.

!   The user must write the subroutine that fills the coordinate arrays
!   and their bounds with actual data.  The following line is simply a
!   a place-holder for the user's code, which should replace it.

!  *** possible user-written call ***

call read_coords(alats, alons, plevs, bnds_lat, bnds_lon)

! Specify path where tables can be found and indicate that existing
!    netCDF files should not be overwritten.

error_flag = cmor_setup(inpath='Test', netcdf_file_action='replace')

! Define dataset as output from the GICC model (first member of an
!   ensemble of simulations) run under IPCC 2xCO2 equilibrium
!   experiment conditions, and provide information to be included as
!   attributes in all CF-netCDF files written as part of this dataset.

mycal = '360_day'

error_flag = cmor_dataset_json("Test/CMOR_input_example.json")


!  Define all axes that will be needed

ilat = cmor_axis(  &
      table='Tables/CMIP6_Amon.json',    &
      table_entry='latitude',       &
      units='degrees_north',        &
      length=lat,                   &
      coord_vals=alats,             &
      cell_bounds=bnds_lat)

ilon2 = cmor_axis(  &
      table='Tables/CMIP6_Lmon.json',    &
      table_entry='longitude',      &
      length=lon,                   &
      units='degrees_east',         &
      coord_vals=alons,             &
      cell_bounds=bnds_lon)

ilat2 = cmor_axis(  &
      table='Tables/CMIP6_Lmon.json',    &
      table_entry='latitude',       &
      units='degrees_north',        &
      length=lat,                   &
      coord_vals=alats,             &
```

```
      cell_bounds=bnds_lat)

ilon = cmor_axis(  &
      table='Tables/CMIP6_Amon.json',    &
      table_entry='longitude',      &
      length=lon,                   &
      units='degrees_east',         &
      coord_vals=alons,             &
      cell_bounds=bnds_lon)

ipres = cmor_axis(  &
      table='Tables/CMIP6_Amon.json',    &
      table_entry='plev19',       &
      units='Pa',                   &
      length=lev2,                   &
      coord_vals=plevs)

!   note that the time axis is defined next, but the time coordinate
!   values and bounds will be passed to cmor through function
!   cmor_write (later, below).

itim = cmor_axis(  &
      table='Tables/CMIP6_Amon.json',    &
      table_entry='time',           &
      units='days since 2030-1-1',  &
      length=ntimes,                &
      interval='20 minutes')
itim2 = cmor_axis(  &
      table='Tables/CMIP6_Lmon.json',    &
      table_entry='time',           &
      units='days since 2030-1-1',  &
      length=ntimes,                &
      interval='20 minutes')

!  define model eta levels (although these must be provided, they will
!    actually be replaced by a+b before writing the netCDF file)
zlevs = (/ 0.1, 0.3, 0.55, 0.7, 0.9 /)
zlev_bnds=(/ 0.,.2, .42, .62, .8, 1. /)

ilev = cmor_axis(  &
      table='Tables/CMIP6_Amon.json',    &
      table_entry='standard_hybrid_sigma',       &
      units='1', &
      length=lev,                   &
      coord_vals=zlevs,             &
      cell_bounds=zlev_bnds)

!   define z-factors needed to transform from model level to pressure
p0 = 1.e5
a_coeff = (/ 0.1, 0.2, 0.3, 0.22, 0.1 /)
```

```
b_coeff = (/ 0.0, 0.1, 0.2, 0.5, 0.8 /)

a_coeff_bnds=(/0.,.15, .25, .25, .16, 0./)
b_coeff_bnds=(/0.,.05, .15, .35, .65, 1./)

error_flag = cmor_zfactor(  &
      zaxis_id=ilev,                    &
      zfactor_name='p0',                &
      units='Pa',                       &
      zfactor_values = p0)

error_flag = cmor_zfactor(  &
      zaxis_id=ilev,                     &
      zfactor_name='b',                  &
      axis_ids= (/ ilev /),              &
      zfactor_values = b_coeff,          &
      zfactor_bounds = b_coeff_bnds  )

error_flag = cmor_zfactor(  &
      zaxis_id=ilev,                     &
      zfactor_name='a',                  &
      axis_ids= (/ ilev /),              &
      zfactor_values = a_coeff,          &
      zfactor_bounds = a_coeff_bnds )

zfactor_id = cmor_zfactor(  &
      zaxis_id=ilev,                       &
      zfactor_name='ps',                   &
      axis_ids=(/ ilon, ilat, itim /),     &
      units='Pa' )

!  Define the only field to be written that is a function of model level
!    (appearing in IPCC table A1c)

var3d_ids(1) = cmor_variable(    &
      table='Tables/CMIP6_Amon.json',  &
      table_entry=entry3d(1),     &
      units=units3d(1),           &
      axis_ids=(/ ilon, ilat, ilev, itim /),  &
      missing_value=1.0e28, &
      original_name=varin3d(1))

!  Define variables appearing in IPCC table A1c that are a function of pressure
!         (3-d variables)

DO m=2,n3d
    var3d_ids(m) = cmor_variable(    &
        table='Tables/CMIP6_Amon.json',  &
        table_entry=entry3d(m),     &
        units=units3d(m),           &
```

```
        axis_ids=(/ ilon, ilat, ipres, itim /), &
        missing_value=1.0e28,       &
        original_name=varin3d(m))
ENDDO


!  Define variables appearing in IPCC table A1a (2-d variables)

DO m=1,n2d
    if (m.ne.3) then
    var2d_ids(m) = cmor_variable(    &
        table='Tables/CMIP6_Amon.json',      &
        table_entry=entry2d(m),     &
        units=units2d(m),           &
        axis_ids=(/ ilon, ilat, itim /), &
        missing_value=1.0e28,       &
        positive=positive2d(m),     &
        original_name=varin2d(m))
else
    var2d_ids(m) = cmor_variable(    &
        table='Tables/CMIP6_Lmon.json',      &
        table_entry=entry2d(m),     &
        units=units2d(m),           &
        axis_ids=(/ ilon2, ilat2, itim2 /), &
        missing_value=1.0e28,       &
        positive=positive2d(m),     &
        original_name=varin2d(m))
endif
ENDDO

PRINT*, ' '
PRINT*, 'completed everything up to writing output fields '
PRINT*, ' '

!  Loop through history files (each containing several different fields,
!       but only a single month of data, averaged over the month).  Then
!       extract fields of interest and write these to netCDF files (with
!       one field per file, but all months included in the loop).

time_loop: DO it=1, ntimes

    ! In the following loops over the 3d and 2d fields, the user-written
    ! subroutines (read_3d_input_files and read_2d_input_files) retrieve
    ! the requested IPCC table A1c and table A1a fields and store them in
    ! data3d and data2d, respectively.  In addition a user-written code
    ! (read_time) retrieves the time and time-bounds associated with the
    ! time sample (in units of 'days since 1970-1-1', consistent with the
    ! axis definitions above).  The bounds are set to the beginning and
    ! the end of the month retrieved, indicating the averaging period.
```

```
  ! The user must write a code to obtain the times and time-bounds for
  !   the time slice.  The following line is simply a place-holder for
  !   the user's code, which should replace it.

call read_time(it, time(1), bnds_time)

call read_3d_input_files(it, varin3d(1), data3d)

error_flag = cmor_write(                                  &
      var_id        = var3d_ids(1),                       &
      data          = data3d,                             &
      ntimes_passed = 1,                                  &
      time_vals     = time,                               &
      time_bnds     = bnds_time   )

call read_2d_input_files(it, varin2d(4), data2d)

error_flag = cmor_write(                                  &
      var_id        = zfactor_id,                         &
      data          = data2d,                             &
      ntimes_passed = 1,                                  &
      time_vals     = time,                               &
      time_bnds     = bnds_time,                          &
      store_with    = var3d_ids(1) )

! Cycle through the 3-d fields (stored on pressure levels),
! and retrieve the requested variable and append each to the
! appropriate netCDF file.

DO m=2,n3d

    ! The user must write the code that fills the arrays of data
    ! that will be passed to CMOR.  The following line is simply a
    ! a place-holder for the user's code, which should replace it.

    call read_3d_input_files(it, varin3d(m), data3d)

    ! append a single time sample of data for a single field to
    ! the appropriate netCDF file.
    error_flag = cmor_write(                                  &
          var_id        = var3d_ids(m),                       &
          data          = data3d,                             &
          ntimes_passed = 1,                                  &
          time_vals     = time,                               &
          time_bnds     = bnds_time  )

    IF (error_flag < 0) THEN
        ! write diagnostic messages to standard output device
        write(*,*) ' Error encountered writing IPCC Table A1c ' &
            // 'field ', entry3d(m), ', which I call ', varin3d(m)
```

```
          write(*,*) ' Was processing time sample: ', time

      END IF

    END DO

    ! Cycle through the 2-d fields, retrieve the requested variable and
    ! append each to the appropriate netCDF file.

    DO m=1,n2d

      ! The user must write the code that fills the arrays of data
      ! that will be passed to CMOR.  The following line is simply a
      ! a place-holder for the user's code, which should replace it.

      call read_2d_input_files(it, varin2d(m), data2d)

      ! append a single time sample of data for a single field to
      ! the appropriate netCDF file.

      error_flag = cmor_write(                                  &
            var_id        = var2d_ids(m),                       &
            data          = data2d,                             &
            ntimes_passed = 1,                                  &
            time_vals     = time,                               &
            time_bnds     = bnds_time  )

      IF (error_flag < 0) THEN
          ! write diagnostic messages to standard output device
          write(*,*) ' Error encountered writing IPCC Table A1a ' &
              // 'field ', entry2d(m), ', which I call ', varin2d(m)
          write(*,*) ' Was processing time sample: ', time

      END IF

    END DO

END DO time_loop

!   Close all files opened by CMOR.

error_flag = cmor_close()

print*, ' '
print*, '*****************************'
print*, ' '
print*, 'ipcc_test_code executed to completion '
print*, ' '
print*, '*****************************'
```

```
END PROGRAM ipcc_test_code
```

# C example

## CMOR user input

- CMOR_input_example.json
  (https://github.com/PCMDI/cmor/blob/main/Test/CMOR_input_example.json)

**Click to expand JSON file**

```
{
    "#note":           "explanation of what source_type is goes here",
    "source_type":            "AOGCM ISM AER",

    "#note":                  "CMIP6 valid experiment_ids are found in CMIP6_C
V.json",
    "experiment_id":          "piControl-withism",
    "activity_id":            "ISMIP6",
    "sub_experiment_id":      "none",

    "realization_index":      "3",
    "initialization_index":   "1",
    "physics_index":          "1",
    "forcing_index":          "1",

    "#note":                  "Text stored in attribute variant_info (recommende
d, not required description of run variant)",
    "run_variant":            "3rd realization",

    "parent_experiment_id":   "historical",
    "parent_activity_id":     "CMIP",
    "parent_source_id":       "PCMDI-test-1-0",
    "parent_variant_label":   "r3i1p1f1",

    "parent_time_units":      "days since 1850-01-01",
    "branch_method":          "standard",
    "branch_time_in_child":   59400.0,
    "branch_time_in_parent":  59400.0,

    "#note":                  "institution_id must be registered at https://gith
ub.com/WCRP-CMIP/CMIP6_CVs/issues/new ",
    "institution_id":         "PCMDI",

    "#note":                  "source_id (model name) must be registered at http
s://github.com/WCRP-CMIP/CMIP6_CVs/issues/new ",
    "source_id":              "PCMDI-test-1-0",

    "calendar":               "360_day",

    "grid":                   "native atmosphere regular grid (3x4 latxlon)",
    "grid_label":             "gn",
    "nominal_resolution":     "10000 km",

    "license":                 "CMIP6 model data produced by Lawrence Livermore
PCMDI is licensed under a Creative Commons Attribution ShareAlike 4.0 Internatio
nal License (https://creativecommons.org/licenses). Consult https://pcmdi.llnl.g
ov/CMIP6/TermsOfUse for terms of use governing CMIP6 output, including citation
requirements and proper acknowledgment. Further information about this data, inc
luding some limitations, can be found via the further_info_url (recorded as a gl
obal attribute in this file) and at https:///pcmdi.llnl.gov/. The data producer
```

```
s and data providers make no warranty, either express or implied, including, bu
t not limited to, warranties of merchantability and fitness for a particular pur
pose. All liabilities arising from the supply of the information (including any
liability arising in negligence) are excluded to the fullest extent permitted b
y law.",

    "#output":                "Root directory for output (can be either a relati
ve or full path)",
    "outpath":                "CMIP6",

    "#note":                  " **** The following descriptors are optional and
may be set to an empty string ",

    "contact ":               "Python Coder (coder@a.b.c.com)",
    "history":                "Output from archivcl_A1.nce/giccm_03_std_2xCO2_22
56.",
    "comment":                "",
    "references":             "Model described by Koder and Tolkien (J. Geophy
s. Res., 2001, 576-591).  Also see http://www.GICC.su/giccm/doc/index.html.  Th
e ssp245 simulation is described in Dorkey et al. '(Clim. Dyn., 2003, 323-35
7.)'",

    "#note":                  " **** The following will be obtained from the CV
and do not need to be defined here",

    "sub_experiment":         "none",
    "institution":            "",
    "source":                 "PCMDI-test 1.0 (1989)",

    "#note":                  " **** The following are set correctly for CMIP6 a
nd should not normally need editing",

    "_controlled_vocabulary_file": "CMIP6_CV.json",
    "_AXIS_ENTRY_FILE":         "CMIP6_coordinate.json",
    "_FORMULA_VAR_FILE":        "CMIP6_formula_terms.json",
    "_cmip6_option":           "CMIP6",

    "mip_era":                "CMIP6",
    "parent_mip_era":         "CMIP6",

    "tracking_prefix":        "hdl:21.14100",
    "_history_template":       "%s ;rewrote data to be consistent with <activit
y_id> for variable <variable_id> found in table <table_id>.",

    "#output_path_template":   "Template for output path directory using tables
keys or global attributes, these should follow the relevant data reference synta
x",
    "output_path_template":    "<mip_era><activity_id><institution_id><source_i
d><experiment_id><_member_id><table><variable_id><grid_label><version>",
    "output_file_template":    "<variable_id><table><source_id><experiment_i
```

```
d><_member_id><grid_label>",
}
```

## C source code

- ipcc_test_code.c (https://github.com/PCMDI/cmor/blob/main/Test/ipcc_test_code.c)
- reader_2D_3D.h (https://github.com/PCMDI/cmor/blob/main/Test/reader_2D_3D.h)

**Click to expand C code**

```
#include <time.h>
#include <stdio.h>
#include<string.h>
#include "cmor.h"
#include <stdlib.h>

void read_coords(alats, alons, plevs, bnds_lat, bnds_lon, lon, lat, lev)
double *alats, *alons;
int *plevs;
double *bnds_lat, *bnds_lon;
int lon, lat, lev;
{
    int i;

    for (i = 0; i < lon; i++) {
        alons[i] = i * 360. / lon;
        bnds_lon[2 * i] = (i - 0.5) * 360. / lon;
        bnds_lon[2 * i + 1] = (i + 0.5) * 360. / lon;
    };

    for (i = 0; i < lat; i++) {
        alats[i] = (lat - i) * 10;
        bnds_lat[2 * i] = (lat - i) * 10 + 5.;
        bnds_lat[2 * i + 1] = (lat - i) * 10 - 5.;
    };

    plevs[0] = 1000;
    plevs[1] = 925;
    plevs[2] = 850;
    plevs[3] = 700;
    plevs[4] = 600;
    plevs[5] = 500;
    plevs[6] = 400;
    plevs[7] = 300;
    plevs[8] = 250;
    plevs[9] = 200;
    plevs[10] = 150;
    plevs[11] = 100;
    plevs[12] = 70;
    plevs[13] = 50;
    plevs[14] = 30;
    plevs[15] = 20;
    plevs[16] = 10;
    plevs[17] = 5;
    plevs[18] = 1;
}

void read_time(it, time, time_bnds)
int it;
double time[];
```

```
double time_bnds[];
{
    time[0] = (it - 0.5) * 30.;
    time_bnds[0] = (it - 1) * 30.;
    time_bnds[1] = it * 30.;

    time[0] = it;
    time_bnds[0] = it;
    time_bnds[1] = it + 1;

}

#include "reader_2D_3D.h"

int main()
/*   Purpose:   To serve as a generic example of an application that */
/*       uses the "Climate Model Output Rewriter" (CMOR) */
/*    CMOR writes CF-compliant netCDF files. */
/*    Its use is strongly encouraged by the IPCC and is intended for use  */
/*       by those participating in many community-coordinated standard  */
/*       climate model experiments (e.g., AMIP, CMIP, CFMIP, PMIP, APE, */
/*       etc.) */
/*   Background information for this sample code: */
/*      Atmospheric standard output requested by IPCC are listed in  */
/*   tables available on the web.  Monthly mean output is found in */
/*   tables A1a and A1c.  This sample code processes only two 3-d  */
/*   variables listed in table A1c ("monthly mean atmosphere 3-D data"  */
/*   and only four 2-d variables listed in table A1a ("monthly mean  */
/*   atmosphere + land surface 2-D (latitude, longitude) data").  The  */
/*   extension to many more fields is trivial. */
/*      For this example, the user must fill in the sections of code that  */
/*   extract the 3-d and 2-d fields from his monthly mean "history"  */
/*   files (which usually contain many variables but only a single time  */
/*   slice).  The CMOR code will write each field in a separate file, but  */
/*   many monthly mean time-samples will be stored together.  These  */
/*   constraints partially determine the structure of the code. */
/*   Record of revisions: */
/*       Date        Programmer(s)           Description of change */
/*       ====        ==========              ==================== */
/*      10/22/03     Rusty Koder              Original code */
/*       1/28/04     Les R. Koder             Revised to be consistent */
/*                                            with evolving code design */
{
    /* ------------------------------- */
    /*   dimension parameters: */
    /* ------------------------------- */
#define   ntimes  2             /* number of time samples to process */
#define   lon  4                /* number of longitude grid cells   */
#define   lat  3                /* number of latitude grid cells */
#define   lev  19               /* number of standard pressure levels */
```

```
#define   n2d  4               /* number of IPCC Table A1a fields to be outpu
t. */
#define   n3d 3                /* number of IPCC Table A1c fields to be outpu
t. */

    /*   Tables associating the user's variables with IPCC standard output  */
    /*   variables.  The user may choose to make this association in a  */
    /*   different way (e.g., by defining values of pointers that allow him  */
    /*   to directly retrieve data from a data record containing many  */
    /*   different variables), but in some way the user will need to map his  */
    /*   model output onto the Tables specifying the MIP standard output. */

    /* ---------------------------------- */

    /* My variable names for IPCC Table A1c fields */
    char varin3d[n3d][6] = { "CLOUD", "U", "T" };

    /* Units appropriate to my data */
    char units3d[n3d][6] = { "%", "m s-1", "K" };

    /* Corresponding IPCC Table A1c entry (variable name)  */
    char entry3d[n3d][3] = { "cl", "ua", "ta" };

    /* My variable names for IPCC Table A1a fields */
    char varin2d[n2d][9] = { "LATENT", "TSURF", "SOIL_WET", "PSURF" };

    /* Units appropriate to my data */
    char units2d[n2d][7] = { "W m-2", "K", "kg m-2", "Pa" };

    char positive2d[n2d][4] = { "down", " ", " ", " " };

    /* Corresponding IPCC Table A1a entry (variable name)  */
    char entry2d[n2d][6] = { "hfls", "tas", "mrsos", "ps" };

/*  uninitialized variables used in communicating with CMOR: */
/*  ------------------------------------------------------- */

    int error_flag;
    int znondim_id, zfactor_id;
    int var2d_ids[n2d];
    int var3d_ids[n3d];
    double data2d[lat * lon];
    double data3d[lev * lat * lon];
    double alats[lat];
    double alons[lon];
    int ilats[lat];
    int ilons[lon];
    double plevs[lev];
    int iplevs[lev];
    long lplevs[lev];
```

```
    float fplevs[lev];
    double Time[2];
    double bnds_time[4];
    double bnds_lat[lat * 2];
    double bnds_lon[lon * 2];
    double zlevs[lev];
    double zlev_bnds[lev + 1];

    double a_coeff[lev] = { 0.1, 0.2, 0.3, 0.22, 0.1 };
    double b_coeff[lev] = { 0.0, 0.1, 0.2, 0.5, 0.8 };
    float p0 = 1.e5;
    double a_coeff_bnds[lev + 1] = { 0., .15, .25, .25, .16, 0. };
    double b_coeff_bnds[lev + 1] = { 0., .05, .15, .35, .65, 1. };
    int ilon, ilat, ipres, ilev, itim;
    double dtmp, dtmp2;

    /*  Other variables: */
    /*  -------------------- */

    int it, m, i, ierr, j;
    int myaxes[10];
    int myaxes2[10];
    int myvars[10];
    char id[CMOR_MAX_STRING];
    char units[CMOR_MAX_STRING];
    char interval[CMOR_MAX_STRING];
    char anames[25][CMOR_MAX_STRING];
    char type;
    char regions[5][23] =
      { "atlantic_arctic_ocean", "indian_pacific_ocean", "pacific_ocean",
        "global_ocean", "sf_bay"
    };
    double timestest[5];
    /* Externals funcs */
    int tables[5];
    char msg[555];
    double bt = 0.;
    /* ============================== */
    /*  Execution begins here: */
    /* ============================== */

    /* Read coordinate information from model into arrays that will be passed
*/
    /*   to CMOR. */
    /* Read latitude, longitude, and pressure coordinate values into  */
    /*   alats, alons, and plevs, respectively.  Also generate latitude and  */
    /*   longitude bounds, and store in bnds_lat and bnds_lon, respectively. */
    /*   Note that all variable names in this code can be freely chosen by */
    /*   the user. */
```

```
/*   The user must write the subroutine that fills the coordinate arrays  */
/*   and their bounds with actual data.  The following line is simply a */
/*   a place-holder for the user's code, which should replace it. */

/*  *** possible user-written call *** */

m = CMOR_EXIT_ON_MAJOR;
j = CMOR_REPLACE_4;
i = 1;
it = 0;
printf("ok mode is:%i\n", m);
ierr = cmor_setup(NULL, &j, NULL, &m, NULL, &i);    //,"  ipcc_test.LOG  ");

read_coords(&alats[0], &alons[0], &iplevs[0], &bnds_lat[0], &bnds_lon[0],
            lon, lat, lev);
int tmpmo[12];
printf("Test code: ok init cmor\n");
char c1[CMOR_MAX_STRING];
char c2[CMOR_MAX_STRING];
strcpy(c1, "GICCM1(2002)\0");
strcpy(c2, "Nat\0");

printf("yep: %s, %s\n", c1, c2);
ierr = cmor_dataset_json("Test/CMOR_input_example.json");

printf("Test code: ok load cmor table(s)\n");
ierr = cmor_load_table("Tables/CMIP6_Omon.json", &tables[0]);
ierr = cmor_load_table("Tables/CMIP6_Amon.json", &tables[1]);

strcpy(id, "time");
strcpy(units, "months since 1980");
strcpy(interval, "1 month");

read_time(0, &Time[0], &bnds_time[0]);
read_time(1, &Time[1], &bnds_time[2]);
ierr =
  cmor_axis(&myaxes[0], id, units, ntimes, &Time[0], 'd', &bnds_time[0], 2,
            interval);

strcpy(id, "latitude");
strcpy(units, "degrees_north");
strcpy(interval, "");
ierr =
  cmor_axis(&myaxes[1], id, units, lat, &alats, 'd', &bnds_lat, 2,
            interval);

strcpy(id, "longitude");
strcpy(units, "degrees_east");
ierr =
  cmor_axis(&myaxes[2], id, units, lon, &alons, 'd', &bnds_lon, 2,
```

```
                interval);

    strcpy(id, "plev19");
    strcpy(units, "hPa");
    ierr =
      cmor_axis(&myaxes[3], id, units, lev, &iplevs, 'i', NULL, 0, interval);

    zlevs[0] = 0.1;
    zlevs[1] = 0.3;
    zlevs[2] = 0.5;
    zlevs[3] = 0.72;
    zlevs[4] = 0.9;

    zlev_bnds[0] = 0.;
    zlev_bnds[1] = .2;
    zlev_bnds[2] = .42;
    zlev_bnds[3] = .62;
    zlev_bnds[4] = .8;
    zlev_bnds[5] = 1.;
/*   p0 = 1.e5; */
/*   a_coeff = { 0.1, 0.2, 0.3, 0.22, 0.1 }; */
/*   b_coeff = { 0.0, 0.1, 0.2, 0.5, 0.8 }; */

/*   a_coeff_bnds={0.,.15, .25, .25, .16, 0.}; */
/*   b_coeff_bnds={0.,.05, .15, .35, .65, 1.}; */

    ierr =
      cmor_axis(&myaxes[4], "standard_hybrid_sigma", "1", 5, &zlevs, 'd',
                &zlev_bnds, 1, interval);

    cmor_set_table(tables[0]);
    /* ok here we declare a "regions" axis */
    printf("Test code: defining axis region \n");
    ierr =
      cmor_axis(&myaxes[5], "basin", "", 4, &regions[0], 'c', NULL, 23,
                interval);

    printf("Test code: Redefining time/lat from 0 table\n");

    strcpy(id, "time");
    strcpy(units, "months since 1980");
    strcpy(interval, "1 month");
    read_time(0, &Time[0], &bnds_time[0]);
    read_time(1, &Time[1], &bnds_time[2]);
    ierr =
      cmor_axis(&myaxes[7], id, units, ntimes, &Time[0], 'd', &bnds_time[0], 2,
                interval);

    strcpy(id, "latitude");
    strcpy(units, "degrees_north");
```

```
    strcpy(interval, "");
    ierr =
      cmor_axis(&myaxes[8], id, units, lat, &alats, 'd', &bnds_lat, 2,
                interval);

    cmor_set_table(tables[1]);

    dtmp = -999;
    dtmp2 = 1.e-4;
    myaxes2[0] = myaxes[0];
    myaxes2[1] = myaxes[3];
    myaxes2[2] = myaxes[1];
    myaxes2[3] = myaxes[2];

    printf("Test code: defining variables from table 1, %s\n", positive2d[0]);
    ierr =
      cmor_variable(&myvars[0], entry2d[0], units2d[0], 3, myaxes, 'd', &dtmp,
                    &dtmp2, positive2d[0], varin2d[0], "no history",
                    "no future");
    ierr =
      cmor_variable(&myvars[1], entry3d[2], units3d[2], 4, myaxes2, 'd', NULL,
                    &dtmp2, NULL, varin3d[2], "no history", "no future");

    printf("Test code: definig tas\n");
    ierr =
      cmor_variable(&myvars[5], "tas", "K", 3, myaxes, 'd', NULL, &dtmp2, NULL,
                    "TS", "no history", "no future");

    myaxes2[1] = myaxes[4];
    ierr =
      cmor_variable(&myvars[2], entry3d[0], units3d[0], 4, myaxes2, 'd', NULL,
                    &dtmp2, NULL, varin3d[0], "no history", "no future");
    ierr =
      cmor_zfactor(&myvars[3], myaxes2[1], "p0", "Pa", 0, NULL, 'f', &p0, NULL);
    ierr =
      cmor_zfactor(&myvars[3], myaxes2[1], "b", "", 1, &myaxes2[1], 'd',
                   &b_coeff, &b_coeff_bnds);
    ierr =
      cmor_zfactor(&myvars[3], myaxes2[1], "a", "", 1, &myaxes2[1], 'd',
                   &a_coeff, &a_coeff_bnds);
/*   printf("defining ap\n"); */
/*   for(i=0;i<5;i++) {a_coeff[i]*=1.e3;printf("sending acoef: %i, %lf\n",i,a_co
eff[i]);} */
/*   for(i=0;i<6;i++) {a_coeff_bnds[i]*=1.e5;printf("sending acoef: %i, %lf\
n",i,a_coeff_bnds[i]);} */
/*   ierr = cmor_zfactor(&myvars[3],myaxes2[1],"ap","hPa",1,&myaxes2[1],'d',&a_c
oeff,&a_coeff_bnds); */
    ierr =
      cmor_zfactor(&myvars[3], myaxes2[1], "ps", "hPa", 3, &myaxes[0], 'd',
                   NULL, NULL);
```

```
    /* ok here we decalre a variable for region axis testing */
    cmor_set_table(tables[0]);
    myaxes2[0] = myaxes[7];     /* time */
    myaxes2[1] = myaxes[5];     /* region */
    myaxes2[2] = myaxes[8];     /* latitudes */
    printf("Test code: ok we define hfogo positive: %s\n", positive2d[0]);
    ierr =
      cmor_variable(&myvars[4], "htovgyre", "W", 3, myaxes2, 'd', NULL, &dtmp2,
                    NULL, varin2d[0], "no history", "no future");

    cmor_set_table(tables[1]);

    for (i = 0; i < ntimes; i++) {
        printf("Test code: writing time: %i of %i\n", i + 1, ntimes);

        printf("2d\n");
        read_2d_input_files(i, varin2d[0], &data2d, lat, lon);
        sprintf(id, "%i", i);
        ierr = cmor_write(myvars[0], &data2d, 'd', NULL, 1, NULL, NULL, NULL);
        if (ierr)
            return (1);
        printf("3d\n");
        read_3d_input_files(i, varin3d[2], &data3d, lev, lat, lon);
        ierr = cmor_write(myvars[1], &data3d, 'd', NULL, 1, NULL, NULL, NULL);
        if (ierr)
            return (1);

        printf("writing tas\n");
        read_2d_input_files(i, varin2d[1], &data2d, lat, lon);
        ierr = cmor_write(myvars[5], &data2d, 'd', NULL, 1, NULL, NULL, NULL);
        if (ierr)
            return (1);

        printf("3d zfactor\n");
        read_3d_input_files(i, varin3d[0], &data3d, 5, lat, lon);
        ierr = cmor_write(myvars[2], &data3d, 'd', NULL, 1, NULL, NULL, NULL);
        if (ierr)
            return (1);

        printf("writing ps\n");
        read_2d_input_files(i, varin2d[3], &data2d, lat, lon);
        ierr = cmor_write(myvars[3], &data2d, 'd', NULL, 1, NULL, NULL, &myvar
s[2]);
        if (ierr)
            return (1);

        /* rereading hfls to fake hfogo */
        printf("2d region\n");
        read_2d_input_files(i, "htov", &data2d, lat, lon);
```

```
        ierr = cmor_write(myvars[4], &data2d, 'd', NULL, 1, NULL, NULL, NULL);
        if (ierr)
            return (1);

    }
    ierr = cmor_close_variable(myvars[0], NULL, NULL);
    ierr = cmor_close();
    return (0);
}
```

# Controlled Vocabulary (CMIP6)

CMIP6 Controlled vocabulary minimum requirements.

- CMOR 3 required a new Controlled Vocabulary file which must contains 4 mandatory keys for CMIP6.
    - institutions_ids: A dictionary of of registered institution IDs with a description.
    - source_ids: A dictionary of registered source IDS (model) with a `specific` description.
    - experiment_ids: A dictionary of experiment_ids (CMIP6) pointing to a dictionary of `specific` metadata.
    - grid_labels: A dictionary of grid labels(gr, gn, ...) pointing to a native_resolution for the selected grid.

**Click to expand example JSON file**

```
{
"CV": {
    "institution_ids": { "BNU":"GCESS, BNU, Beijing, China" },
    "source_ids": { "CESM1-CAM5": "CESM1 (CAM5): model version ca. 2009" },
    "experiment_ids": { "piControl": { } },
    "grid_labels": { "gr":     { "native_resolution":"5 km" } }
   }
}
```

To register, activities, sources or institutions

- Contact: cmor@listserv.llnl.gov

CMIP6 required global attributes

- CMIP6_CV.json (https://github.com/PCMDI/cmor/blob/main/TestTables/CMIP6_CV.json)

**Click to expand example JSON section**

```
    "required_global_attributes":
        [
        "variant_label",
        "activity_id",
        "branch_method",
        "Conventions",
        "creation_date",
        "mip_era",
        "data_specs_version",
        "experiment_id",
        "experiment",
        "forcing_index",
        "further_info_url",
        "frequency",
        "grid",
        "grid_label",
        "native_resolution",
        "initialization_index",
        "institution",
        "institution_id",
        "license",
        "physics_index",
        "product",
        "realization_index",
        "realm",
        "variant_label",
        "source",
        "source_id",
        "source_type",
        "sub_experiment",
        "sub_experiment_id",
        "table_id",
        "tracking_id",
        "variable_id"
        ],
```

- CMOR validates required attributes using list of values or regular expression(REGEX)

**Click to expand example JSON section**

```
    "required_parent_attributes": [
        "parent_experiment_id"
        ],

    "variant_label": [ "^r[[:digit:]]\\{1,\\}i[[:digit:]]\\{1,\\}p[[:digi
t:]]\\{1,\\}f[[:digit:]]\\{1,\\}$" ],

    "sub_experiment_id": [ "^s[[:digit:]]\\{4,4\\}$", "none" ],

    "product": [ "output" ] ,

    "mip_era": [ "CMIP6" ],

    "further_info_url": [ "http://furtherinfo.es-doc.org/[[:alpha:]]\\{1,\\}" ],
```


## Registered activities

**Click to expand example JSON section**


```
"activity_id":[
            "AerChemMIP",
            "C4MIP",
            "CFMIP",
            "CMIP",
            "CORDEX",
            "DAMIP",
            "DCPP",
            "DynVarMIP",
            "FAFMIP",
            "GMMIP",
            "GeoMIP",
            "HighResMIP",
            "ISMIP6",
            "LS3MIP",
            "LUMIP",
            "OMIP",
            "PMIP",
            "RFMIP",
            "SIMIP",
            "ScenarioMIP",
            "VIACSAB",
            "VolMIP"
],
```


## Registered sources

**Click to expand example JSON section**

```
    "source_ids": {
        "ACCESS1-0": "ACCESS1.0: adaptation of unified model with interactive ch
emistry (ca. 2012)" ,
        ...
        },
```

### Registered institutions

**Click to expand example JSON section**

```
    "institution_ids": {
            "NSF-DOE-NCAR":"NSF/DOE NCAR (National Center for Atmospheric Resear
ch) Boulder, CO, USA"
            ...
    },
```

### valid grids

**Click to expand example JSON section**

```
    "grid_labels": {

        "gs1x1":     { "native_resolution":"1x1" },
        "gs1x1 gn":  { "native_resolution":"1x1" },
        "gs1x1 gr":  { "native_resolution":"1x1" },
        "gn": { "native_resolution":[  "5 km",   "10 km",   "25 km",   "50 k
m",   "100 km", "250 km",
                "500 km", "1000 km", "2500 km", "5000 km", "10000 km" ] },
        "gr":  { "native_resolution":[  "5 km",   "10 km",   "25 km",   "50 k
m",   "100 km", "250 km",
                "500 km", "1000 km", "2500 km", "5000 km", "10000 km" ] }

    },
```

### Registered experiments

**Click to expand example JSON section**

```
experiment_ids": {

            "piControl":{
                "activity_id":[
                    "CMIP"
                ],
                "additional_allowed_model_components":[
                    "AER",
                    "CHEM",
                    "BGC"
                ],
                "description":"DECK: control",
                "end_year":"",
                "experiment":"pre-industrial control",
                "experiment_id":"piControl",
                "min_number_yrs_per_sim":"500",
                "parent_activity_id":[
                    "CMIP"
                ],
                "parent_experiment_id":[
                    "piControl-spinup"
                ],
                "required_model_components":[
                    "AOGCM"
                ],
                "start_year":"",
                "sub_experiment_id":[
                    "none"
                ],
                "tier":"1"
}
```

# CMIP6 Table Excerpt

## WCRP Data Request links

The names of the MIP tables are constructed using the tokens described below, and a frequency token. The frequency is generally taken from the CIP6 frequency vocabulary, except for the “monClim” frequency, which is abbreviated to “clim”.

- MIP Tables in the CMIP6 Data Request

about mip table (https://earthsystemcog.org/projects/wip/mip_table_about)

- **WCRP Table List**

  Here is the WCRP mip tables list (http://clipc-services.ceda.ac.uk/dreq/index/miptable.html)

- **Cmor Tables**

  Corresponding CMOR3 CMIP6 tables to link above can be found here (https://github.com/PCMDI/cmip6-cmor-tables/tree/master/Tables).

## Header

```
"Header": {
    "mip_era":     "CMIP6",
    "approx_interval": "30.00000",
    "realm": "atmos",
    "product": "output",
    "cmor_version": "3.3",
    "Conventions": "CF-1.6 CMIP-6.0",
    "table_id": "Table Amon",
    "data_specs_version": "3.0",
    "generic_levels": "alevel alevhalf",
    "missing_value": "1e20",
    "table_date": "01 April 2016"
},
```

axis_entry

```
"axis_entry": {
    "forecast": {
        "stored_direction": "increasing",
        "must_have_bounds": "no",
        "long_name": "ensemble time axis",
        "standard_name": "time",
        "out_name": "forecast",
        "type": "double",
        "units": "days since 1900-01-01",
        "value": "0.0",
        "axis": "T"
    },
    "plev17": {
        "requested": [
            "100000.",
            "92500.",
            "85000.",
            "70000.",
            "60000.",
            "50000.",
            "40000.",
            "30000.",
            "25000.",
            "20000.",
            "15000.",
            "10000.",
            "7000.",
            "5000.",
            "3000.",
            "2000.",
            "1000."
        ],
        "stored_direction": "decreasing",
        "z_factors": "",
        "positive": "down",
        "must_have_bounds": "no",
        "valid_min": "",
        "requested_bounds": "",
        "z_bounds_factors": "",
        "bounds_values": "",
        "long_name": "pressure",
        "standard_name": "air_pressure",
        "value": "",
        "out_name": "plev",
        "type": "double",
        "units": "Pa",
        "formula": "",
        "climatology": "",
        "tolerance": "0.001",
        "valid_max": "",
```

```
        "axis": "Z"
    },
    "height2m": {
        "requested": "",
        "stored_direction": "increasing",
        "z_factors": "",
        "positive": "up",
        "must_have_bounds": "no",
        "valid_min": "1.0",
        "requested_bounds": "",
        "z_bounds_factors": "",
        "bounds_values": "",
        "long_name": "height",
        "standard_name": "height",
        "value": "2.0",
        "out_name": "height",
        "type": "double",
        "units": "m",
        "formula": "",
        "climatology": "",
        "tolerance": "",
        "valid_max": "10.0",
        "axis": "Z"
    },
    "latitude": {
        "requested": "",
        "stored_direction": "increasing",
        "z_factors": "",
        "positive": "",
        "must_have_bounds": "yes",
        "valid_min": "-90.0",
        "requested_bounds": "",
        "z_bounds_factors": "",
        "bounds_values": "",
        "long_name": "latitude",
        "standard_name": "latitude",
        "value": "",
        "out_name": "lat",
        "type": "double",
        "units": "degrees_north",
        "formula": "",
        "climatology": "",
        "tolerance": "",
        "valid_max": "90.0",
        "axis": "Y"
    },
    "longitude": {
        "requested": "",
        "stored_direction": "increasing",
        "z_factors": "",
```

```
        "positive": "",
        "must_have_bounds": "yes",
        "valid_min": "0.0",
        "requested_bounds": "",
        "z_bounds_factors": "",
        "bounds_values": "",
        "long_name": "longitude",
        "standard_name": "longitude",
        "value": "",
        "out_name": "lon",
        "type": "double",
        "units": "degrees_east",
        "formula": "",
        "climatology": "",
        "tolerance": "",
        "valid_max": "360.0",
        "axis": "X"
    },
    "time": {
        "requested": "",
        "stored_direction": "increasing",
        "z_factors": "",
        "positive": "",
        "must_have_bounds": "yes",
        "valid_min": "",
        "requested_bounds": "",
        "z_bounds_factors": "",
        "bounds_values": "",
        "long_name": "time",
        "standard_name": "time",
        "value": "",
        "out_name": "time",
        "type": "double",
        "units": "days since ?",
        "formula": "",
        "climatology": "",
        "tolerance": "",
        "valid_max": "",
        "axis": "T"
    }
},
```

variable_entry

```
"variable_entry": {
    "rsutcs": {
        "comment": "",
        "dimensions": "longitude latitude time",
        "frequency": "mon",
        "positive": "up",
        "valid_min": "0",
        "long_name": "TOA Outgoing Clear-Sky Shortwave Radiation",
        "standard_name": "toa_outgoing_shortwave_flux_assuming_clear_sky",
        "modeling_realm": "atmos",
        "cell_measures": "time: mean",
        "cell_methods": "area: areacella",
        "ok_min_mean_abs": "54.7",
        "units": "W m-2",
        "out_name": "rsutcs",
        "type": "real",
        "valid_max": "444",
        "ok_max_mean_abs": "73.36"
    },
    "tas": {
        "comment": "near-surface (usually, 2 meter) air temperature",
        "dimensions": "longitude latitude time height2m",
        "frequency": "mon",
        "positive": "",
        "valid_min": "180.6",
        "long_name": "Near-Surface Air Temperature",
        "standard_name": "air_temperature",
        "modeling_realm": "atmos",
        "cell_measures": "time: mean",
        "cell_methods": "area: areacella",
        "ok_min_mean_abs": "262.4",
        "units": "K",
        "out_name": "tas",
        "type": "real",
        "valid_max": "335.1",
        "ok_max_mean_abs": "293"
    },
    "tasforecast": {
        "comment": "near-surface (usually, 2 meter) air temperature",
        "dimensions": "longitude latitude time height2m forecast",
        "frequency": "mon",
        "positive": "",
        "valid_min": "180.6",
        "long_name": "Near-Surface Air Temperature",
        "standard_name": "air_temperature",
        "modeling_realm": "atmos",
        "cell_measures": "time: mean",
        "cell_methods": "area: areacella",
        "ok_min_mean_abs": "262.4",
        "units": "K",
```

```
        "out_name": "tas",
        "type": "real",
        "valid_max": "335.1",
        "ok_max_mean_abs": "293"
    },
    "rldscs": {
        "comment": "",
        "dimensions": "longitude latitude time",
        "frequency": "mon",
        "positive": "down",
        "valid_min": "33.55",
        "long_name": "Surface Downwelling Clear-Sky Longwave Radiation",
        "standard_name": "surface_downwelling_longwave_flux_in_air_assuming_clea
r_sky",
        "modeling_realm": "atmos",
        "cell_measures": "time: mean",
        "cell_methods": "area: areacella",
        "ok_min_mean_abs": "238.6",
        "units": "W m-2",
        "out_name": "rldscs",
        "type": "real",
        "valid_max": "543.6",
        "ok_max_mean_abs": "293.8"
    },
    "n2oglobal": {
        "comment": "",
        "dimensions": "time",
        "frequency": "mon",
        "positive": "",
        "valid_min": "",
        "long_name": "Global Mean Mole Fraction of N2O",
        "standard_name": "mole_fraction_of_nitrous_oxide_in_air",
        "modeling_realm": "atmos atmosChem",
        "cell_measures": "time: mean",
        "cell_methods": "",
        "ok_min_mean_abs": "",
        "units": "1e-09",
        "out_name": "n2oglobal",
        "type": "real",
        "valid_max": "",
        "ok_max_mean_abs": ""
    },
    "ts": {
        "comment": "'skin' temperature (i.e., SST for open ocean)",
        "dimensions": "longitude latitude time",
        "frequency": "mon",
        "positive": "",
        "valid_min": "176.8",
        "long_name": "Surface Temperature",
        "standard_name": "surface_temperature",
```

```
        "modeling_realm": "atmos",
        "cell_measures": "time: mean",
        "cell_methods": "area: areacella",
        "ok_min_mean_abs": "262.8",
        "units": "K",
        "out_name": "ts",
        "type": "real",
        "valid_max": "339.6",
        "ok_max_mean_abs": "293.3"
    },
    "clt": {
        "comment": "cloud area fraction",
        "dimensions": "longitude latitude time",
        "frequency": "mon",
        "positive": "",
        "valid_min": "-0.0001822",
        "long_name": "Total Cloud Fraction",
        "standard_name": "cloud_area_fraction_in_atmosphere_layer",
        "modeling_realm": "atmos",
        "cell_measures": "time: mean",
        "cell_methods": "area: areacella",
        "ok_min_mean_abs": "39.37",
        "units": "1.0",
        "out_name": "clt",
        "type": "real",
        "valid_max": "105",
        "ok_max_mean_abs": "84.98"
    },
    "tasmax": {
        "comment": "maximum near-surface (usually, 2 meter) air temperature (ad
d cell_method attribute 'time: max')",
        "dimensions": "longitude latitude time height2m",
        "frequency": "mon",
        "positive": "",
        "valid_min": "181.9",
        "long_name": "Daily Maximum Near-Surface Air Temperature",
        "standard_name": "air_temperature",
        "modeling_realm": "atmos",
        "cell_measures": "time: maximum within days time: mean over days",
        "cell_methods": "area: areacella",
        "ok_min_mean_abs": "264.9",
        "units": "K",
        "out_name": "tasmax",
        "type": "real",
        "valid_max": "341.9",
        "ok_max_mean_abs": "294"
    }
}
```

# CMIP6 Global Attributes

## CMIP6 Global Attributes

- *variant_label*
- *activity_id*
- *branch_method*
- *Conventions*
- *creation_date*
- *mip_era*
- *data_specs_version*
- *experiment_id*
- *experiment*
- *forcing_index*
- *further_info_url*
- *frequency*
- *grid*
- *grid_label*
- *nominal_resolution*
- *initialization_index*
- *institution*
- *institution_id*
- *license*
- *physics_index*
- *product*
- *realization_index*
- *realm*
- *variant_label*
- *source*
- *source_id*
- *source_type*

- *sub_experiment*
- *sub_experiment_id*
- *table_id*
- *tracking_id*
- *variable_id*

# CMIP6 User Input

## Notes

1. Keys beginning with character _ will not be written in netCDF file as attribute. They can be use for template filename of template path.
2. Keys beginning with character **#** can be used as comment.

## CMIP6 CMOR User Input

CMIP6_global_attributes_filenames_CVs.doc
(https://docs.google.com/document/d/1h0r8RZr_f3-8egBMMh7aqLwy3snpD6_MrDz1q8n5XUk)

- *_controlled_vocabulary_file*:"Specify Controlled Vocabulary file name – default: CMIP6_CV.json"
- *_AXIS_ENTRY_FILE*: "Specify Coordinate table file(axes) – default: CMIP6_coordinate.json"
- *_FORMULA_VAR_FILE*: "Speciry Formula terms table file – defalut: CMIP6_formula_terms.json"
- *_cmip6_option*: "used to trigger validation for CMIP6 only."
- *activity_id*: "Specify an activity PMIP, GeoMIP"
- *output*: "Output Path where files are written – must be created by the user."
- *experiment_id*: "Correspond to id found in "_controlled_vocabulary_file""
- *source_type*: "type of model used",
- *sub_experiment*: "description of sub-experiment",
- *sub_experiment_id*: "none",
- *parent_sub_experiment_id*:
- *parent_mip_era*:
- *mip_era*:
- *institution*:
- *source*:
- *calendar*:
- *realization_index*:
- *initialization_index*:
- *physics_index*:
- *forcing_index*:

- *contact *:
- *history*:
- *comment*:
- *references*:
- *institution_id*:
- *model_id*:
- *forcing*:
- *parent_variant_label*:
- *parent_experiment_id*:
- *branch_time*:
- *parent_activity_id*:
- *parent_source_id*:
- *branch_method*:
- *branch_time_in_child*:
- *branch_time_in_parent*:
- *branch_time_units_in_parent*:
- *further_info_url*: “http://furtherinfo.es-doc.org/<mip_era>/<institution_id><source_id><experiment_id><sub_experiment_id><variant_label>”,
- *grid*:
- *grid_label*:
- *nominal_resolution*:
- *run_variant*:
- *source_id*:
- *output_path_template*: “<mip_era><activity_id><institution_id><source_id><experiment_id><member_id><table><variable_id><grid_
- *version*: Set this to name the <version> part of the output path template. If not set, then CMOR will use the current date as the version in the format of “vYYYYMMDD” (ex. “v20220718”),
- *output_file_template*: “<variable_id><table><source_id><experiment_id><member_id><grid_label>[<time_range>]”,
- *license*: Use the following template
    - “CMIP6 model data produced by **<institution’s name>** is licensed under a **<license>**. Consult https://pcmdi.llnl.gov/CMIP6/TermsOfUse for terms of use governing CMIP6 output, including citation requirements and proper acknowledgment. Further information about this data, including some limitations, can be found via the further_info_url (recorded as a global attribute in this file)**[ and at <URL>]**. The data

producers and data providers make no warranty, either express or implied, including, but not limited to, warranties of merchantability and fitness for a particular purpose. All liabilities arising from the supply of the information (including any liability arising in negligence) are excluded to the fullest extent permitted by law."

- **[ and at <URL>]** is a placeholder for an optional URL for information on the data.
- Choose one of the following for **<license>**
    - Creative Commons Attribution 4.0 International License (https://creativecommons.org/licenses/by/4.0/)
    - Creative Commons CC0 1.0 Universal Public Domain Dedication License (https://creativecommons.org/publicdomain/zero/1.0/)

# Appendix A

## Critical Errors

The following errors are considered as CRITICAL and will cause a CMOR code to stop.

1. Calling a CMOR function before running cmor_setup
2. NetCDF version is neither 3.6.3 or 4.1 or greater
3. Udunits could not parse units
4. Incompatible units
5. Udunits could not create a converter
6. Logfile could not be open for writing
7. Output directory does not exist
8. Output directory is not a directory
9. User does not have read/write privileges on the output directory
10. Wrong value for error_mode
11. wrong value for netCDF mode
12. error reading udunits system
13. NetCDF could not set variable attribute
14. Dataset does not have one of the required attributes (required attributes can be defined in the MIP table)
15. Required global attribute is missing
16. If CMIP5 project: source attributes does not start with model_id attribute.
17. Forcing dataset attribute is not valid
18. Leap_year defined with invalid leap_month
19. Invalid leap month (<1 or >12)
20. Leap month defined but no leap year
21. Negative realization number
22. Zfactor variable not defined when needed
23. Zfactor defined w/o values and NOT time dependent.
24. Variable has axis defined with formula terms depending on axis that are not part of the variable
25. NetCDF error when creating zfactor variable

26. NetCDF Error defining compression parameters
27. Calling cmor_write with an invalid variable id
28. Could not create path structure
29. “variable id” contains a “_” or a ‘-‘ this means bad MIP table.
30. “file_suffix” contains a “_”
31. Could not rename the file you’re trying to append to.
32. Trying to write an “Associated variable” before the variable itself
33. Output file exists and you’re not in append/replace mode
34. NetCDF Error opening file for appending
35. NetCDF could not find time dimension in a file onto which you want to append
36. NetCDF could not figure out the length time dimension in a file onto which you want to append
37. NetCDF could not find your variable while appending to a file
38. NetCDF could not find time dimension in the variable onto which you’re trying to append
39. NetCDF could not find time bounds in the variable onto which you’re trying to append
40. NetCDF mode got corrupted.
41. NetCDF error creating file
42. NetCDF error putting file in definition mode
43. NetCDF error writing file global attribute
44. NetCDF error creating dimension in file
45. NetCDF error creating variable
46. NetCDF error writing variable attribute
47. NetCDF error setting chunking parameters
48. NetCDF error leaving definition mode
49. Hybrid coordinate, could not find “a” coefficient
50. Hybrid coordinate, could not find “b” coefficient
51. Hybrid coordinate, could not find “a_bnds” coefficient
52. Hybrid coordinate, could not find “b_bnds” coefficient
53. Hybrid coordinate, could not find “p0” coefficient
54. Hybrid coordinate, could not find “ap” coefficient
55. Hybrid coordinate, could not find “ap_bnds” coefficient
56. Hybrid coordinate, could not find “sigma” coefficient
57. Hybrid coordinate, could not find “sigma_bnds” coefficient

58. NetCDF writing error
59. NetCDF error closing file
60. Could not rename temporary file to its final name.
61. Cdms could not convert time values for calendar.
62. Variable does not have all required attributes (cmor_variable)
63. Reference variable is defined with “positive”, user did not pass it to cmor_variable
64. Could not allocate memory for zfactor elements
65. Udunits error freeing units
66. Udunits error freeing converter
67. Could not allocate memory for zfactor_bounds
68. Calling cmor_variable before reading in a MIP table
69. Too many variable defined (see appendix on CMOR limits)
70. Could not find variable in MIP table
71. Wrong parameter “positive” passed
72. No “positive” parameter passed to cmor_variable and it is required for this variable
73. Variable defined with too many (not enough) dimensions
74. Variable defined with axis that should not be on this variable
75. Variable defined within existing axis (wrong axis_id)
76. Defining variable with axes defined in a MIP table that is not the current one.
77. Defining a variable with too many axes (see annex on CMOR limits)
78. Defining variable with axes ids that are not valid.
79. Defining variable with grid id that is not valid.
80. Defining a variable with dimensions that are not part of the MIP table (except for var named “latitude” and “longitude”, since they could have grid axes defined in another MIP table)
81. Trying to retrieve length of time for a variable defined w/o time length
82. Trying to retrieve variable shape into an array of wrong rank (Fortran only really)
83. Calling cmor_write with time values for a timeless variable
84. Cannot allocate memory for temporary array to write
85. Invalid absolute mean for data written (lower or greater by one order of magintudethan what the MIP table allows)
86. Calling cmor_write with time values when they have already been defined with cmor_axis when creating time axis
87. Cannot allocate memory to store time values

88. Cannot allocate memory to store time bounds values
89. Time values are not monotonic
90. Calling cmor_write w/o time values when no values were defined via cmor_axis when creating time axis
91. Time values already written in file
92. Time axis units do not contain “since” word (cmor_axis)
93. Invalid data type for time values (ok are ‘f’,’l’,’i’,’d’)
94. Time values are not within time bounds
95. Non monotonic time bounds
96. Longitude axis spread over 360 degrees.
97. Overlapping bound values (except for climatological data)
98. bounds and axis values are not stored in the same order
99. requested value for axis not present
100. approximate time axis interval much greater (>20%) than the one defined in your MIP table
101. calling cmor_axis before loading a MIP table
102. too many axes defined (see appendix on CMOR limits)
103. could not find reference axis name in current MIP table
104. output axis needs to be standard_hybrid_sigma and input axis is not one of : “standard_hybrid_sigma”, “alternate_hybrid_sigma”, “standard_sigma”
105. MIP table requires to convert axis to unknown type
106. requested “region” not present on axis
107. axis (with bounds) values are in invalid type (valid are: ‘f’,’d’,’l’,’i’)
108. requested values already checked but stored internally, could be bad user cleanup
109. MIP table defined for version of CMOR greater than the library you’re using
110. too many experiments defined in MIP table (see appendix on CMOR limits)
111. cmor_set_table used with invalid table_id
112. MIP table has too many axes defined in it (see appendix on CMOR limits)
113. MIP table has too many variables defined in it (see appendix on CMOR limits)
114. MIP table has too many mappings defined in it (see appendix on CMOR limits)
115. MIP table defines the same mapping twice
116. grid mapping has too many parameters (see appendix on CMOR limits)
117. grid has different number of axes than what grid_mapping prescribes.
118. Could not find all the axes required by grid_mapping

119. Call to cmor_grid with axis that are not created yet via cmor_axis
120. Too many grids defined (see appendix on cmor_limits)
121. Call to cmor_grid w/o latitude array
122. Call to cmor_grid w/o longitude array

# Appendix B

## Limits in cmor

The following are defined in cmor.h

#define CMOR_MAX_STRING 1024

#define CMOR_DEF_ATT_STR_LEN 256

#define CMOR_MAX_ELEMENTS 500

#define CMOR_MAX_AXES CMOR_MAX_ELEMENTS*3

#define CMOR_MAX_VARIABLES CMOR_MAX_ELEMENTS

#define CMOR_MAX_GRIDS 100

#define CMOR_MAX_DIMENSIONS 7

#define CMOR_MAX_ATTRIBUTES 100

#define CMOR_MAX_ERRORS 10

#define CMOR_MAX_TABLES 10

#define CMOR_MAX_GRID_ATTRIBUTES 25

# Contact us

### CMOR3 issues

https://github.com/PCMDI/cmor/issues (https://github.com/PCMDI/cmor/issues)

### CMIP6 table issues

https://github.com/PCMDI/cmip6-cmor-tables/issues (https://github.com/PCMDI/cmip6-cmor-tables/issues)

http://dreq01.vanillaforums.com/categories/cmip6-issues
(http://dreq01.vanillaforums.com/categories/cmip6-issues)

### CMOR3 documentations issues

https://github.com/PCMDI/cmor3_documentation/issues
(https://github.com/PCMDI/cmor3_documentation/issues)

### Mailing list

cmor@lists.llnl.gov

1. To subscribe to the `CMOR` mailing list,
    - send an email writing `subscribe cmor` in the text of your message to LISTSERV@listserv.llnl.gov
2. To learn how to use the `LISTSERV` email application, go to https://www.lsoft.com/resources/manuals.asp (https://www.lsoft.com/resources/manuals.asp)
3. The following email addresses are used for the specified purposes:
    - To send an email to the list: CMOR@LISTS.LLNL.GOV (page 0)
    - To contact the list owner: CMOR-request@LISTSERV.LLNL.GOV
    - To send commands to the `LISTSERV` server: LISTSERV@LISTSERV.LLNL.GOV
    - To unsubscribe from the list: CMOR-signoff-request@LISTSERV.LLNL.GOV
4. INTERACTING WITH LISTSERV BY EMAIL *
    - You may leave the list at any time by sending a `SIGNOFF CMOR` command (include `SIGNOFF CMOR` in the email body to: LISTSERV@LISTSERV.LLNL.GOV)
    - Or by sending a blank email to: CMOR-signoff-request@LISTSERV.LLNL.GOV
    - You can also tell `LISTSERV` how you want it to confirm the receipt of messages that you send to the list. To send yourself a copy of your own messages, send a `SET CMOR REPRO` command.

- Alternatively, to have `LISTSERV` send you a short acknowledgement instead of the entire message, send a `SET CMOR ACK NOREPRO` command. Finally, you can turn off acknowledgements completely with the `SET CMOR NOACK NOREPRO` command.
- This list is available in digest form. If you wish to receive the digested version of the postings, then issue a `SET CMOR DIGEST` command.
- Please note that it is presently possible for other people to determine that you are signed up to the list using the REVIEW command, which returns the email address and name of all the subscribers (include `Review CMOR` in the email body) to: LISTSERV@LISTSERV.LLNL.GOV
- If you do not want your name to be visible, then just issue a `SET CMOR CONCEAL` command.
- Following instructions from the list owner, your subscription options have been set to `"REPRO IETFHDR` ” rather than the usual `LISTSERV` defaults. For more information about subscription options, send a `QUERY CMOR` command to LISTSERV@LISTSERV.LLNL.GOV.