



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Climate Model Output Rewriter (CMOR)

K. E. Taylor, C. Doutriaux, J.-Y. Peterschmitt

June 10, 2004

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Climate Model Output Rewriter (CMOR)

Karl E. Taylor, Charles Doutriaux, and Jean-Yves Peterschmitt

11 May 2004

Design Considerations and Overview

This document describes a software library called "Climate Model Output Rewriter" (CMOR),¹ which comprises a set of FORTRAN 90 functions that can be used to produce CF-compliant² netCDF³ 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"⁴ and include, for example, AMIP, CMIP, CFMIP, PMIP, APE, and IPCC 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 data. 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 with FORTRAN, 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:

- For data that are a function of longitude and latitude, only grids representable as a Cartesian product of longitude and latitude axes are allowed. Model output on other grids, such as "thin" grids, grids with rotated poles, and irregular grids, must

¹ CMOR is pronounced "C-more", which suggests that CMOR should enable a wide community of scientists "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 insrutable 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 See <http://www.cgd.ucar.edu/cms/eaton/cf-metadata>

³ See <http://my.unidata.ucar.edu/content/software/netcdf/>

⁴ "MIP" is an acronym for "model intercomparison project".

be mapped to a longitude-latitude Cartesian grid before being passed to CMOR. Most of the MIPs and most diagnostic software also impose this constraint.⁵

- Each file contains a single output variable (along with coordinate 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 efficiently 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 series of files.
- For metadata, different MIPs may have different requirements, but these are accommodated by CMOR, within the constraints of the CF convention.
- 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 ASCII 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.

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).

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`).

⁵ The CMOR capabilities may be extended in the future to support other types of grids, but this would be expected to require a substantial programming effort.

- 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).
- 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 the original data to prepare it to meet MIP requirements. Toward this end, the code allows the following flexibility (with the MIP requirements obtained by CMOR from the appropriate MIP table):

- 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.
- In many cases, 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 can 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 software (see <http://my.unidata.ucar.edu/content/software/udunits/>), then CMOR performs the transformation before writing the data. Otherwise, CMOR will return an error. To enable this capability, the user will have to obtain udunits and, for time transformations, cdms (??).
- 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 with the MIP requirements (e.g., carries temperature at 1.5 m instead of 2 m), then the user can override the table specifications.

The code does not, however, include a capability to interpolate data, either in the vertical or horizontally. If a user stores data on model levels, but a MIP requests it on standard pressure levels, then the user must interpolate before passing the data to CMOR. Similarly, if the data are originally stored on a non-Cartesian longitude-latitude grid, then the user must map the data to a Cartesian grid before passing it 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`, identifying the project, experiment, and table.
- `Conventions`, ('CF-1.0')
- `history`, including any user-provided history along with a "timestamp" generated by CMOR and a statement that the data conform with both the CF standards and those of a particular MIP.

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

- `institution`, identifying the modeling center contributing the output.
- `source`, identifying the model version that generated the output.
- `contact`, providing the name and email of someone responsible for the data
- `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.

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

- `calendar`, (when appropriate) indicating the calendar type assumed by the model.

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.
- `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.

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

- `original_name`, containing the name of the variable as it is known at the user's home institution.
- `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 appropriate 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.

Acknowledgements

Several individuals have supported the development of this 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. Finally, we appreciate the encouragement expressed by the WGCM for developing CMOR.

Description of CMOR Functions

Note: 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 are 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 provides MIP-specific default values for some parameters. All arguments not in brackets and not followed by an equal sign are always required.

```
error_flag = cmor_setup(inpath='./', netcdf_file_action='preserve', set_verbosity=2,  
exit_control=2)
```

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] = path to directory where the needed MIP-specific tables reside.

[netcdf_file_action] = controls handling of existing netCDF files. If the value passed is 'replace', any existing file with the same name as the one CMOR is trying to create will be overwritten; if the file does not exist, it will be created. If value is 'append', an existing file will be appended; if the file does not exist, it will be created. If value is 'preserve', the program will error exit if the file exists.

[set_verbosity] controls how informational messages and error messages generated by CMOR are handled. Only the most important messages will be sent to a "summary log." All messages will be sent to the "detailed log" and also, if set_verbosity = 2, they will be sent to the standard output device (typically the user's screen). If set_verbosity = 1, only error messages will be sent to standard output, and if set_verbosity = 0, then all CMOR output to standard output will be suppressed. This optional argument is ignored in the beta release of CMOR.

[exit_control] determines if errors will trigger program to exit (0=never stop; 1=stop only if severe errors; 2=stop even after minor errors detected). This optional argument is ignored in the beta release of CMOR.

returns: a negative integer if an error is encountered; otherwise returns 0.

```
error_flag = cmor_dataset(outpath, experiment_id, institution, source, calendar,  
realization=1, [contact], [history], [comment], [references], [leap_year],  
[leap_month], [month_lengths])
```

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`.

Arguments:

`outpath` = path where all output files in this dataset will be written (including both model output netCDF files and log and error files). The user should never put into a single subdirectory output from more than one experiment (e.g., AMIP and CMIP) or more than one model (e.g., two different models from the same modeling center) or more than one realization of an ensemble of simulations. It is also suggested that in most cases the user refrain from placing output defined by different tables in the same directory. An example of a reasonably descriptive directory structure that would be acceptable is: '[pathroot]/CMIP_1percent/member_1/table_1'. The directories in this path indicate the experiment (1% per year increase of CO₂), the member of the ensemble of simulations that is stored there (the first member), and the CMIP table name. (Note that if output listed in more than 1 table of the dataset will be written in a single pass, then it would be inappropriate to include the table name in the path.)

`experiment_id` = character string identifying the experiment within the project that generated the data (e.g., 'control', 'perturbation', etc.) See individual MIP home pages for the official experiment designations (or see the MIP-table list of "expt_id_ok" acceptable i.d.'s).

`institution` = character string identifying the institution that generated the data [e.g., 'NCAR (National Center for Atmospheric Research, Boulder, CO, USA)']

`source` = character string identifying the model version as it is referred to in public talks. Additionally, this attribute must include the year (i.e., model vintage) when this model version was first used in a scientific application. Finally, it should include information concerning the component models. The following template should be used in constructing this string: '[model_name] [year] atmosphere: [model_name] ([technical_name], [resolution_and_levels]); ocean: [model_name] ([technical_name], [resolution_and_levels]); sea ice: [model_name] ([technical_name]); land: [model_name] ([technical_name])' As an example, "source" might contain the string: 'CCSM2 2002 atmosphere: CAM2 (cam2_0_brnchT_itea_2, T42L26); ocean: POP (pop2_0_ver_1.4.3, 2x3L15); sea ice: CSIM4; land: CLM2.0'. For some MIP's it might be appropriate to list only a single component, in which case the descriptor (e.g., 'atmosphere') may be omitted along with the other model components (e.g., 'CAM2 2002 (cam2_0_brnchT_itea_2, T42L26)'). Additional explanatory information may follow the required information.

`calendar` = CF-compliant calendar specification (e.g., 'gregorian', 'noleap', etc.) This argument must be included even in the case of a non-standard calendar, in which case it must not be given one of the calendars currently defined by CF ('gregorian', 'standard', 'proleptic_gregorian', 'noleap', '365_day', '360_day', 'julian', and 'none'), and it must not be completely

blank or a null string. It would be acceptable, for example, to assign 'non_standard' to this argument in the case of a non-standard calendar.

[realization] = an integer distinguishing among members of an ensemble of simulations (e.g., 1, 2, 3, etc.). If only a single simulation was performed, then this argument should be given the value 1 (which is also the default value).

[contact] = name and contact information (e.g., email, address, phone number) of person who should be contacted for more information about the data.

[history] = audit trail for modifications to the original data, each modification typically preceded by a "timestamp". The "history" attribute provided here will be a global one and should not depend on which variable is contained in the file. A variable-specific "history" can also be included in calling `cmor_variable`, described below.

[comment] = miscellaneous information about the data or methods used to produce it. Each MIP may encourage the user to provide different information here. For example, the user may be asked to include a description of how the initial conditions for a simulation were specified and how the model was spun-up (including the length of the spin-up period).

[references] = Published or web-based references that describe the data or methods used to produce it. Typically, the user should provide references describing the model formulation here.

[leap_year] = for non-standard calendars (otherwise omit), an integer, indicating an example of a leap year.

[leap_month] = for non-standard calendars (otherwise omit), an integer in the range 1-12, specifying which month is lengthened by a day in leap years (1=January).

[month_lengths] = for non-standard calendars (otherwise omit), an integer vector of size 12, specifying the number of days in the months from January through December (in a non-leap year).

returns: a negative integer if an error is encountered; otherwise returns 0.

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

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. 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.

Arguments:

`table` = character string containing the filename of the MIP-specific table where the axis defined here appears. (e.g., 'IPCC_table_A1', 'AMIP_table_1a', 'AMIP_table_2', 'CMIP_table_2', etc.)

`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 may differ from the units of the coordinates written to the netCDF file by CMOR. For non-standard calendars (e.g., models with no leap year), conversion of time values can be made only if CMOR is built with CDMS.) . 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 length of the dimension. This argument is required except when either the declared size of `coord_vals` is equal to `length` or `coord_vals` is absent. As noted below, `coord_vals` may be absent only for a time coordinate, in which case the user may choose to indicate with `[length]` the number of time samples that will eventually be written by CMOR before closing the file. This will allow CMOR to determine whether all the time-samples the user intended to write were in fact written.

`[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 for a time coordinate, in which case the user may optionally pass the coordinate values when the `cmor_write` function is called. If the time coordinate values will be passed when the `cmor_write` function is called, the `coord_vals` argument must be omitted when `cmor_axis` is called to define the time axis. Note that the values must be ordered monotonically (for non-character strings), 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.

`[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.

`[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: a negative integer if an error is encountered; otherwise returns a positive integer uniquely identifying the axis being written.

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

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:

`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 (as it appears in the MIP table) that will be defined by this function.

`[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 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.

`[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='none'`, or set `units='1'`.

`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 or set zfactor_values='none'.

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'.

returns: a negative integer if an error is encountered; otherwise returns a positive integer uniquely identifying the z-factor being written.

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

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 cmor_variable function will typically be repeatedly invoked to define other variables.

Arguments:

table = character string containing the filename of the MIP-specific table where table_entry (described next) can be found (e.g., 'IPCC_table_A1', 'AMIP_table_1a', 'AMIP_table_2', 'CMIP_table_2', etc.)

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='percent'; for a fraction, units='1'; for parts per million, units='1e-6', etc.).

axis_ids = 1-d array containing integers returned by cmor_axis, which specifies the axes associated with the variable that this function defines. These i.d.'s should be ordered consistently with the data that will be passed to CMOR through function cmor_write (see documentation below). Scalar ("singleton") dimensions defined in the MIP table may be omitted, if they have not been explicitly redefined by the user through calls to cmor_axis. If the size of the 1-d array is larger than the number of dimensions, the

'unused' dimension i.d.'s must be set to 0. Note that if a single axis is passed, it must not be passed as a scalar but as a rank 1 array of length 1.

[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 either 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 $\pm \text{tolerance} * \text{missing_value}$ 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 fluxes and surface stress, and 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: a negative integer if an error is encountered; otherwise returns a positive integer uniquely identifying the variable being written.

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

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), any dimension of size (i.e., length) 1 may or may not be included in determining the rank. 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 may be declared as either 2 or 3. Likewise for surface air temperature, which has a singleton dimension indicating that the height is 2 meters, that dimension may be omitted from the data array. 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). In any case the dimension sizes (lengths) must not be smaller than those expected.

`[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). If one wanted to break up the time-sequence of *tas* fields into several files (each containing one or more time samples), the user might choose to use the suffix to indicate which years were stored in each file (e.g., *tas_A1_1979-1988*, *tas_A1_1989-1998*, etc.). Alternatively (and more simply) the user might simply use the suffix to number the files sequentially (e.g., *tas_A1_1*, *tas_A1_2*, etc.). There are no restrictions on the suffix except that it must yield unique filenames. 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.

`[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). The units should be consistent with those passed as an argument to cmor_axis in defining the time axis.

[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) 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 only when writing a z-factor. (See description of the zfactor function above.)

returns: a negative integer if an error is encountered; otherwise returns 0.

```
error_flag = cmor_close(var_id=0)
```

Description: Close a single file specified by optional argument var_id or if the argument is omitted, close all files created by CMOR (including log files). To be safe, before exiting any program that invokes CMOR, it is safest to call this function with the argument omitted (or set to 0).

Arguments:

[var_id] = a handle (i.e., an integer returned by cmor_variable) identifying an individual variable (in a specific dataset) and the associated output file that will be closed by this function.

returns: a negative integer if an error is encountered; otherwise returns 0.

Sample Program 1

```
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 Ala and Alc. This sample code processes only two 3-d
!       variables listed in table Alc ("monthly mean atmosphere 3-D data"
!       and only four 2-d variables listed in table Ala ("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

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 :: n2d = 4        ! number of IPCC Table Ala fields to be
!                                     output.
INTEGER, PARAMETER :: n3d = 2        ! number of IPCC Table Alc fields to
!                                     be output.

!   Define 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=('/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 = ('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, DIMENSION(n2d) :: var2d_ids
INTEGER, DIMENSION(n3d) :: var3d_ids
REAL, DIMENSION(lon,lat) :: data2d
REAL, DIMENSION(lon,lat,lev) :: data3d
DOUBLE PRECISION, DIMENSION(lat) :: alats
DOUBLE PRECISION, DIMENSION(lon) :: alons
DOUBLE PRECISION, DIMENSION(lev) :: 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

INTEGER :: ilon, ilat, ipres, ilev, itim

! Other variables:
! -----

INTEGER :: it, m

! =====
! Execution begins here:
! =====

! Read coordinate information from model output 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.

! *** call to user-written subroutine ***

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

! Specify path where tables can be found and indicate that existing
! netCDF files should 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.

error_flag = cmor_dataset(
    outpath='Test',
    experiment_id='2xCO2 equilibrium experiment',
    institution=
    'GICC (Generic International Climate Center, ' //
    'Geneva, Switzerland)',
    source='GICCM 2002(giccm_0_brnchT_itea_2, T63L32)',
    calendar='noleap',
    realization=1,
    contact = 'Rusty Koder (koder@middle-earth.net) ',
    history='Output from archive/giccm_03_std_2xCO2_2256.',
    comment='Equilibrium reached after 30-year spin-up ' //
    'after which data were output starting with nominal ' //
    'date of January 2030',
    references='Model described by Koder and Tolkien ' //
    '(J. Geophys. Res., 2001, 576-591). Also ' //
    'see http://www.GICC.su/giccm/doc/index.html ' //
    '2XCO2 simulation described in Dorkey et al. ' //
    '(Clim. Dyn., 2003, 323-357.)' )

! Define all axes that will be needed

ilat = cmor_axis( &
    table='IPCC_table_A1',
    table_entry='latitude',
    units='degrees_north',
    length=lat,
    coord_vals=alats,
    cell_bounds=bnds_lat)

ilon = cmor_axis( &
    table='IPCC_table_AA1',
    table_entry='longitude',
    length=lon,
    units='degrees_east',
    coord_vals=alons,
    cell_bounds=bnds_lon)

ipres = cmor_axis( &
    table='IPCC_table_A1',
    table_entry='pressure',
    units='Pa',

```

```

        length=lev,                &
        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='IPCC_table_A1',        &
    table_entry='time',           &
    units='days since 1970-1-1',  &
    length=ntimes,                &
    interval='20 minutes')

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

DO m=1,n3d
    var3d_ids(m) = cmor_variable( &
        table='IPCC_table_A1',    &
        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 Ala (2-d variables)

DO m=1,n2d
    var2d_ids(m) = cmor_variable( &
        table='IPCC_table_A1',    &
        table_entry=entry2d(m),   &
        units=units2d(m),         &
        axis_ids=(/ ilon, ilat, itim /), &
        missing_value=-1.0e28,    &
        positive=positive2d(m),   &
        original_name=varin2d(m))
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 Alc and table Ala 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, bnds_time)

! 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=1,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 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

```

Sample Portion of a MIP Table (which will be made available by MIP organizers to contributing groups)

The user normally need not be concerned with the details contained in this table.

```

cmor_version: 0.8          ! version of CMOR that can read this table
cf_version: 1.0            ! version of CF that output conforms to
project_id: IPCC           ! project id
table_id: Table A1        ! table id
table_date: 7 April 2004 ! date this table was constructed

expt_id_ok: 'committed climate change experiment' ! official name(s) of
expt_id_ok: 'SRES A2 experiment' ! project's experiments
expt_id_ok: 'control experiment (for committed climate change experiment)'
expt_id_ok: '720 ppm stabilization experiment (SRES A1B)'
expt_id_ok: '550 ppm stabilization experiment (SRES B1)'
expt_id_ok: '1%/year CO2 increase experiment (to doubling)'
expt_id_ok: '1%/year CO2 increase experiment (to quadrupling)'
expt_id_ok: 'control experiment (for 1%/year CO2 run)'
expt_id_ok: '2xCO2 equilibrium experiment'
expt_id_ok: 'control experiment (for 2xCO2 run)'

magic_number: -1          ! used to check whether this file has been
                          ! altered from the official version.
                          ! should be set to number of non-blank
                          ! characters in file.
approx_interval: 30.      ! approximate spacing between successive time
                          ! samples (in units of the output time
                          ! coordinate.
missing_value: 1.e20      ! value used to indicate a missing value
                          ! in arrays output by netCDF as 32-bit IEEE
                          ! floating-point numbers (float or real)

!*****
!
! SUBROUTINE ARGUMENT DEFAULT INFORMATION
!
!*****
!
! set default specifications for subroutine arguments to:
!   required/indeterminate/optional/ignored/forbidden
!   (indeterminate may or may not be required information, but is not always
!   required as an argument of the function call)
!
!
!=====
subroutine_entry: cmor_axis
!=====
!
required: table axis_name units length coord_vals cell_bounds
ignored: interval
!
!=====
subroutine_entry: cmor_variable
!=====
!
required: table table_entry units axis_ids
indeterminate: missing_value
optional: tolerance original_name history comment

```



```

ignored: positive
!
!=====
subroutine_entry: cmor_write
!=====
!
required: var_id data
indeterminate: ntimes_passed time_vals time_bnds store_with
optional: file_suffix
!
!*****
!
!   TEMPLATE FOR AXES
!
!*****
!
!=====
!axis_entry:                ! (required)
!=====
!
!   Override default argument specifications for cmor_axis
!-----
!   acceptable arguments include units length coord_vals cell_bounds interval
!required:                ! (default: table axis_name units length
!                           coord_vals cell_bounds)
!indeterminate:
!optional:
!ignored:                ! (default: interval)
!forbidden:
!-----
!
!   Axis attributes:
!-----
!standard_name:            ! (required)
!units:                    ! (required)
!axis:                    ! X, Y, Z, T (default: undeclared)
!positive:                ! up or down (default: undeclared)
!long_name:                ! (default: undeclared)
!-----
!
!   Additional axis information:
!-----
!out_name:                ! (default: same as axis_entry)
!type:                    ! double (default), real, character, integer
!stored_direction:        ! increasing (default) or decreasing
!valid_min:                ! type: double precision (default: no check performed)
!valid_max:                ! type: double precision (default: no check performed)
!requested:                ! space-separated list of requested coordinates
!                           ! (default: undeclared)
!requested_bounds:        ! space-separated list of requested coordinate bounds
!                           ! (default: undeclared)
!tol_on_requests:         ! fractional tolerance for meeting request
!                           ! (default=1.e-3, which is used in the formula:
!                           !     eps = MIN(( tol*interval between grid-points)
!                           !           and (1.e-3*tol*coordinate value)))
!value:                    ! of scalar (singleton) dimension
!bounds_values:           ! of scalar (singleton) dimension bounds
!-----
!
!*****
!
!   TEMPLATE FOR VARIABLES
!

```

```

!*****
!
!=====
!variable_entry:                ! (required)
!=====
!
!      Override default argument specifications for cmor_variable
!-----
!      acceptable arguments include  file_suffix missing_value tolerance
!                                     original_name history comment positive
!required:                        ! (default: table table_entry units axis_ids)
!indeterminate:                  ! (default: file_suffix missing_value)
!optional:                       ! (default: original_name history comment)
!ignored:                        ! (default: positive)
!forbidden:
!-----
!
! Variable attributes:
!-----
!standard_name:                  ! (required)
!units:                          ! (required)
!cell_methods:                  ! (default: undeclared)
!long_name:                     ! (default: undeclared)
!comment:                       ! (default: undeclared)
!-----
!
! Additional variable information:
!-----
!dimensions:                    ! (required) (scalar dimension(s) should appear
!                               ! last in list)
!out_name:                      ! (default: variable_entry)
!type:                          ! real (default), double, integer
!positive:                      ! up or down (default: undeclared)
!valid_min:                    ! type: real (default: no check performed)
!valid_max:                    ! type: real (default: no check performed)
!ok_min_mean_abs:              ! type: real (default: no check performed)
!ok_max_mean_abs:              ! type: real (default: no check performed)
!-----
!
!
!*****
!
!  AXIS INFORMATION
!
!*****
!
!=====
axis_entry: longitude
!=====
!
!-----
!
! Axis attributes:
!-----
standard_name:    longitude
units:            degrees_east
axis:             X
long_name:        longitude
!-----
!
! Additional axis information:
!-----
out_name:         lon

```

```

valid_min:      0.          ! CMOR will add n*360 to input values
                        ! (where n is an integer) to ensure
                        ! longitudes are in proper range.  The
                        ! data will also be rearranged
                        ! appropriately.
valid_max:      360.        ! see above comment.
!-----
!
!
!=====
axis_entry: latitude
!=====
!
! Axis attributes:
!-----
standard_name:   latitude
units:           degrees_north
axis:            Y
long_name:       latitude
!-----
!
! Additional axis information:
!-----
out_name:        lat
valid_min:       -90.
valid_max:       90.
!-----
!
!
!=====
axis_entry: time
!=====
!
!   Override default argument specifications for cmor_axis
!-----
required: interval
indeterminate: coord_vals cell_bounds
!-----
!
! Axis attributes:
!-----
standard_name:   time
units:           days since ?    ! the user's basetime will be used
axis:            T
long_name:       time
!-----
!
!
!=====
axis_entry: pressure
!=====
!
!   Override default argument specifications for cmor_axis
!-----
ignored: cell_bounds
!-----
!
! Axis attributes:
!-----
standard_name:   air_pressure
units:           Pa
axis:            Z
positive:        down

```

```

long_name:      pressure
!-----
!
! Additional axis information:
!-----
out_name:      plev
valid_min:     0.
valid_max:     110000.
requested:     10000. 20000. 30000. 40000. 50000.
!-----
!
!
!=====
axis_entry: height1
!=====
!
! Override default argument specifications for cmor_axis
!-----
ignored: cell_bounds
!-----
!
! Axis attributes:
!-----
standard_name: height
units:         m
axis:          Z
positive:      up
long_name:     height
!-----
!
! Additional axis information:
!-----
out_name:      height
valid_min:     0.
valid_max:     10.
value:         2.
!-----
!
!
!=====
axis_entry: height2
!=====
!
! Override default argument specifications for cmor_axis
!-----
ignored: cell_bounds
!-----
!
! Axis attributes:
!-----
standard_name: height
units:         m
axis:          Z
positive:      up
long_name:     height
!-----
!
! Additional axis information:
!-----
out_name:      height
valid_min:     0.
valid_max:     30.
value:         10.

```

```

!-----
!
!=====
axis_entry: depth1
!=====
!
!-----
!
! Axis attributes:
!-----
standard_name:    depth
units:            m
axis:             Z
positive:         down
long_name:        depth
!-----
!
! Additional axis information:
!-----
out_name:         depth
valid_min:        0.0
valid_max:        1.0
value:            0.05
bounds_values:    0.0 0.1
!-----
!
!
!*****
!
! VARIABLE INFORMATION
!
!*****
!
!=====
variable_entry: tas
!=====
!
! Variable attributes:
!-----
standard_name:    air_temperature
units:            K
cell_methods:     time: mean
long_name:        Surface Air Temperature
!-----
!
! Additional variable information:
!-----
dimensions:       longitude latitude time height1
valid_min:        200.
valid_max:        340.
ok_min_mean_abs:  270.
ok_max_mean_abs:  300.
!-----
!
!
!=====
variable_entry: hfls
!=====
!
! Override default argument specifications for cmor_variable
!-----
required: positive
!-----

```

```

!
! Variable attributes:
!-----
standard_name: upward_surface_latent_heat_flux
units:        W m-2
cell_methods: time: mean
long_name:     Surface Latent Heat Flux
!-----
!
! Additional variable information:
!-----
dimensions:    longitude latitude time
positive:      up
valid_min:     -50.
valid_max:     300.
ok_min_mean_abs: 20.
ok_max_mean_abs: 150.
!-----
!
!
!=====
variable_entry: mrsos
!=====
!
! Variable attributes:
!-----
standard_name: moisture_content_of_soil_layer
units:        kg m-2
cell_methods: time: mean
long_name:     Moisture in Upper 0.1 m of Soil Column
comment:       includes subsurface frozen water but not surface snow and ice
!-----
!
! Additional variable information:
!-----
dimensions:    longitude latitude time depth1
!-----
!
!
!=====
variable_entry: ua
!=====
!
! Variable attributes:
!-----
standard_name: eastward_wind
units:        m s-1
cell_methods: time: mean
long_name:     Zonal Wind Component
!-----
!
! Additional variable information:
!-----
dimensions:    longitude latitude pressure time
valid_min:     -200.
valid_max:     300.
ok_min_mean_abs: 0.1
ok_max_mean_abs: 100.
!-----
!
!
!=====
variable_entry: ta

```

```

!=====
!
! Variable attributes:
!-----
standard_name: air_temperature
units:        K
cell_methods: time: mean
long_name:    Temperature
!-----
!
! Additional variable information:
!-----
dimensions:    longitude latitude pressure time
valid_min:     150.
valid_max:     350.
ok_min_mean_abs: 200.
ok_max_mean_abs: 300.
!-----
!
!
!=====
variable_entry: pr
!=====
!
! Variable attributes:
!-----
standard_name: precipitation
units:        kg m-2 s-1
cell_methods: time: mean
long_name:    Precipitation
comment:      includes all types (rain, snow, large-scale, convective, etc.)
!-----
!
! Additional variable information:
!-----
dimensions:    longitude latitude time
valid_min:     0.0
valid_max:     1.e-4
ok_min_mean_abs: 1.e-6
ok_max_mean_abs: 5.e-5
!-----
!
!
!=====
variable_entry: cl
!=====
!
! Variable attributes:
!-----
standard_name: cloud_area_fraction
units:        %
cell_methods: time: mean
long_name:    Total Cloud Fraction
!-----
!
! Additional variable information:
!-----
dimensions:    longitude latitude zlevel time
valid_min:     0.0
valid_max:     100.0
ok_min_mean_abs: 10.0
ok_max_mean_abs: 90.0
!-----

```

```

!
!
!=====
variable_entry: hfogo
!=====
!
! Variable attributes:
!-----
standard_name: northward_ocean_heat_transport
units: W
cell_methods: time: mean
long_name: Northward Ocean Heat Transport
!-----
!
! Additional variable information:
!-----
dimensions: latitude region time
valid_min: -4.e15
valid_max: 4.e15
ok_min_mean_abs: 0.0
ok_max_mean_abs: 4.e15
!-----
!
!
!=====
variable_entry: ps
!=====
!
! Variable attributes:
!-----
standard_name: surface_air_pressure
units: Pa
cell_methods: time: mean
long_name: Surface Pressure
!-----
!
! Additional variable information:
!-----
dimensions: longitude latitude time
valid_min: 0.5e5
valid_max: 1.2e5
ok_min_mean_abs: 0.9e5.
ok_max_mean_abs: 1.1e5
!-----
!

```