The CDO CMOR operator

Produce CMIP-compliant climate model output
September 2018
CDO 1.9.6 with CMOR 3.4.0

Fabian Wachsmann, Stephanie Legutke, Joerg Wegner, Martin Schupfner

Deutsches Klimarechenzentrum (DKRZ)

Contents

1. Introduction

The Climate Data Operators (**CDO**) software is a collection of operators for standard processing of climate and forecast model data. Here, only the single operator **cdo cmor** is documented. This is necessary because, in contrast to other operators, it includes many configuration options to rearrange model output and to add metadata. It features an interface to the Climate Model Output Rewriter library CMOR developed at PCMDI. This library comprises a set of functions which can be used to produce files that fulfill the requirements of model intercomparison projects (MIPs). The output resulting from CMOR is "self-describing" and facilitates analysis of results across models.

cdo cmor provides an easy interface to CMOR in order to simplify its usage. It comprises all CMOR functions and therefore supports all requested climate data structures including rotated grids, hybrid vertical or character axes. However, the operator itself does not represent a tool for arithmetics or diagnostics. For this purpose, the reader is recommended to use other operators which can be connected with **cdo cmor** in an operator chain to reduce I/O traffic.

In contrast to other CMOR applications developed for one specific project or one specific model, the operator is applicable for output of many climate models including all submodel realms like atmosphere, ocean or sea-ice and for many projects. In fact, necessary configurations for such an unrestricted application needs to be delivered by the user however the required user input can be reduced to a minimum. This is enabled by exploiting the information which is available in the input file and which is retrieved from the climate data interface CDI. This interface provides a fast and file format independent access to the input files. The long term goal is to enable the conversion of output of all earth system models (ESM) participating in CMIP.

Different versions of the CMOR library have been published over the years. A stable operator was developed at the DKRZ and Max-Planck-Institute for Meteorology based on CMOR version 2 which is appropriate to build CMIP5 (phase 5 of the Coupled Model Intercomparison Project) compliant NetCDF model output. The operator was validated with the CMIP5 database generated at the DKRZ.

As a next step, the operator has been prepared to include CMOR version 3 as well so that CMIP6 compliant data can be produced. However, since changes of both, the CMIP6 data standard and request, as well as of CMOR version 3 can not be excluded, the operator may require updates in order to remain compatible with the most recent versions.

The operator will be further enhanced at the DKRZ. Next updates will also focus on the long term goal of facilitating the preparation of project compliant output for future CMIP phases and other projects.

This manual is based on a CDO installation with CMOR version 2. In the following, red boxes highlight aspects where **cdo cmor** differs when applied with recent CMOR 3 versions. The examples given in 'example' sections can be tried with files provided at the HPC of DKRZ, mistral, in `/work/bm0021/cdo_incl_cmor`. Copy the 'example' directory to a local directory. The operations have to be executed in the latter folder in order to use a default info table '.cdocmorinfo' (see 4.1.2).

This manual explains how to create a file which complies to a project data standard, the standard itself is not explained. It is assumed that the user is familiar with the corresponding data standard including the meaning and the vocabulary of the metadata and data requirements. The CMIP6 Participation Guidance for Modelers includes all necessary documents about the CMIP6 data standard.

2. Installation

The installation of CDO with CMOR is complex because a number of packages used by both, CDO and CMOR, need to be combined. It is highly recommended to use the operator on the DKRZ system (mistral) if possible where additional user support can be given.

2.1. DKRZ system

On the login node of mistral, a CDO version 1.7.2 is loaded per default. All CDO versions from version 1.9.4. on are installed with CMOR 3.3 on mistral. A corresponding CDO version can be loaded by typing for example:

```
module unload cdo
module load cdo/1.9.4-gcc64
```

The most recent tag of CDO including cdo cmor operator is linked on mistral in `/work/bm0021/cdo_incl_cmor/cdo_recent_cmor2` for CDOs installed with CMOR2 and `/work/bm0021/cdo_incl_cmor/cdo_recent_cmor3` for CDOs installed with CMOR3. In `/work/bm0021/cdo_incl_cmor`, a script for the installation in a user-defined directory is given.

2.2. Local installation

2.2.1. Using Conda

The most recent CDO version can be easily installed with the most recent CMOR version by using Conda:

```
#update conda
conda update conda
#name the environment for cdo with cmor
cdoenv=cdocmor
#install develop-cdo which contains CMOR "conda-forge/label/dev::cdo"
#use conda-forge channel with "-c conda-forge"
#set $cdoenv as environment with --name
conda create --name ${cdoenv} conda-forge/label/dev::cdo -c conda-forge
#then activate it
source activate ${cdoenv}
```

2.2.2. Using build script

For this purpose it is recommended to use CDO versions greater equal 1.9.4. Tagged versions of the source code are provided at `https://svn.dkrz.de/mad/Model/cmor-support/branches/cdo_incl_cmor` and can be downloaded after a registration on request. Recent tags starting from 06.14.17 support installation with either CMOR version 2.92 or versions greater equal 3.2.3 so that, selectively, CMIP5 and CMIP6 compliant data can be produced. An example script supporting the installation of the downloaded code on a linux computer is given in Appendix C. It requires downloads of other packages. Note that this installation results in full CDO support for GRIB and NETCDF file formats.

3. Usage

This chapter introduces the operator syntax (section 3.1) and describes parameters and arguments as well as their format. A special argument of the operator and the fact that no outfile name specification is allowed are illuminated in section 3.2 and 3.3 respectively.

3.1. Syntax

The **cdo cmor** operator requires at least one parameter and one argument (note that, for most use cases, this is *not* sufficient). The parameter is a *MIP-table* which contains the requested variables and the argument is the input file *infile*. Both files can be specified with a relative or absolute path. In contrast to many other CDO operators, the outfile name can not be specified because it is part of the project standard and accordingly generated by CMOR. The syntax of the operator is:

```
    cdo cmor,MIP-table[,keyvaluelist] infile
```

The *MIP-table* always has to be the first parameter of **cdo cmor**. A second parameter can be specified in format of a comma separated *keyvaluelist*. This list consists of `keyword`=*values* pairs where the *values* can be specified as a comma separated list (CSL):

```
    key=value[,value,...][,key=value[,value,...]]
```

The position of keywords in the list does not matter. All available keywords are given in Appendix A. For some keywords only one value is allowed. Every keyword has a long and a short name version. The short version is composed of the first character of each partial string of the keyword separated by underscores ('_'). For example, the corresponding shortname for keyword `cmor_name` is cn.

3.1.1. Examples

Note that each *MIP-table* and *infile* for all examples are provided at the DKRZ system mistral in `/work/bm0021/cdo_incl_cmor/examples`. With the following command, all variables in example_interface.nc will be converted.

```
    cdo cmor,cmip5_mip_tables/CMIP5_Amon example_interface.nc            #example 1
```

A subset of variables from the input file can be selected for processing by using keyword `cmor_name`. The values must be names of CMOR variables which appear in the *MIP-table* provided as argument 1. If no `cmor_name` is specified, *infile* variable names are interpreted as `cmor_names`. If a specified `cmor_name` does not appear in the *MIP-table* passed as argument 1, the operator aborts.

```
    cdo cmor,cmip5_mip_tables/CMIP5_Amon,cmor_name=tas,uas example_interface.nc #example 2
    cdo cmor,cmip5_mip_tables/CMIP5_Amon,cn=tas,uas example_interface.nc        #example 3
```

#example 3 shows a command variant using the short name version `cn` for `cmor_name`.

The above calls of the operator can only be successful if the input file is a NetCDF file and contains all information necessary for the conversion to a compliant format as global and variable attributes. This minimum information consists of:

- Spatial information i.e. 'longitude' and 'latitude' coordinates of the data grid.
- Temporal information:
  - Time stamps for each time step (relative or absolute)
  - `calendar`
  - `required_time_units`
- All global attributes:
  - `project_id`, `experiment_id`, `source`, `institution`, ...
- Attributes of the variables listed in `cmor_name`:
  - `units`

3.2. MIP-table

The data request of CMIP-like projects is translated into MIP-tables which are readable for CMOR. One MIP-table usually contains the format of variables including variable names (`cmor_names`), variable attributes and required dimensions requested with one specific time cell method, frequency or shape and for one specific realm. The tables are named and distinguished with the help of these parameters. E.g., the monthly requested variable `tas` (Near-surface Air Temperature) is defined in a *MIP-table* named 'Amon' where 'A' stands for Atmosphere and 'mon' for monthly. However, the nomenclature of MIP-tables is not based on a strict rule.

The same `cmor_name` can occur in more than one *MIP-table*, e.g. if a variable is not only requested for monthly frequency but also for daily. This subdivision is of importance since a variable can have different requirements for different frequencies. A **CMOR variable** is the unique combination of a *MIP-table* and a `cmor_name`. The operator must be able to identify the requested **CMOR variable** which is why the MIP-table needs to be given in any call of the operator.

3.3. Why no name specification for the output file?

The output file as well as its name and its path are well-defined and created by CMOR. Name and path are constructed using a subset of meta data. This subset is called Data Reference Syntax (DRS).

For both, name and path, DRS components are arranged according to a template defined by each project. The two control keyword, `drs` and `drs_root` are available to control whether and where this output path is created (see section 4.4). Note that the build-rules of the CMIP6 standard for constructing outfile name and path differ slightly from those of CMIP5. The templates are given below.

CMOR version dependent

CMOR2 sets the file names and paths according to the CMIP5 build-rules. These build-rules can not be modified by the user and, therefore, if CMOR2 is used to rewrite data according to a standard different from that of CMIP5, a renaming of the output paths and files is necessary. One exception is the project CORDEX for which the output file directory structure is also created by the operator compliantly to the project build-rule.

The CMIP5 *output file template*:
"<variable_id>_ <table_id>_<model_id>_<experiment_id>_<member>_*period* "
and the *output path template*:
"<activity_id>/<product>/<institude_id>/<model_id>/
<experiment_id>/<frequency>/<realm>/<variable_id>/<member>/ ".

CMOR3 allows the user to specify the build rules for output file names. The defaults are those of CMIP6:

The CMIP6 *output file template*:
" <variable_id>_ <table_id>_ <source_id>_ <experiment_id>_ <member_id>_ <grid_label>_*period*"
The *output path template*:" <mip_era>/<activity_id>/<institution_id>/<source_id>/
<experiment_id>/<member_id>/<table_id>/<variable_id>/<grid_label>/<version>/".

4. Configuration

In order to highlight the function of **cdo cmor**, it is reasonable to distinguish between the data stream and the meta data stream from the input file into **cdo cmor** and, subsequently, the output file. The operator serves the purpose to *rewrite* the input data and not to manipulate it. By rewriting, the data is restructured: Dimensions may be reordered, netCDF is used as output file format, each output file includes only one variable and the output file will be compressed per default.

Besides two minor data changing processes, the values of the variables will not be modified. These processes are: Units conversion if the input variable units deviate from the project standard units as well as sign conversion if the positive axis of the input variable is directed oppositely to the designated direction of positive values (see also section 4.2).

The meta data and its configuration can be classified into three fields: global attributes, variable attributes, and coordinates and its attributes (approximately temporal and spatial description). **cdo cmor** offers different methods to set the meta data configuration. One valid meta data configuration method is introduced in the example of section 3.1.1: The input file is a netCDF file and includes all meta data required by the project for the target variable.

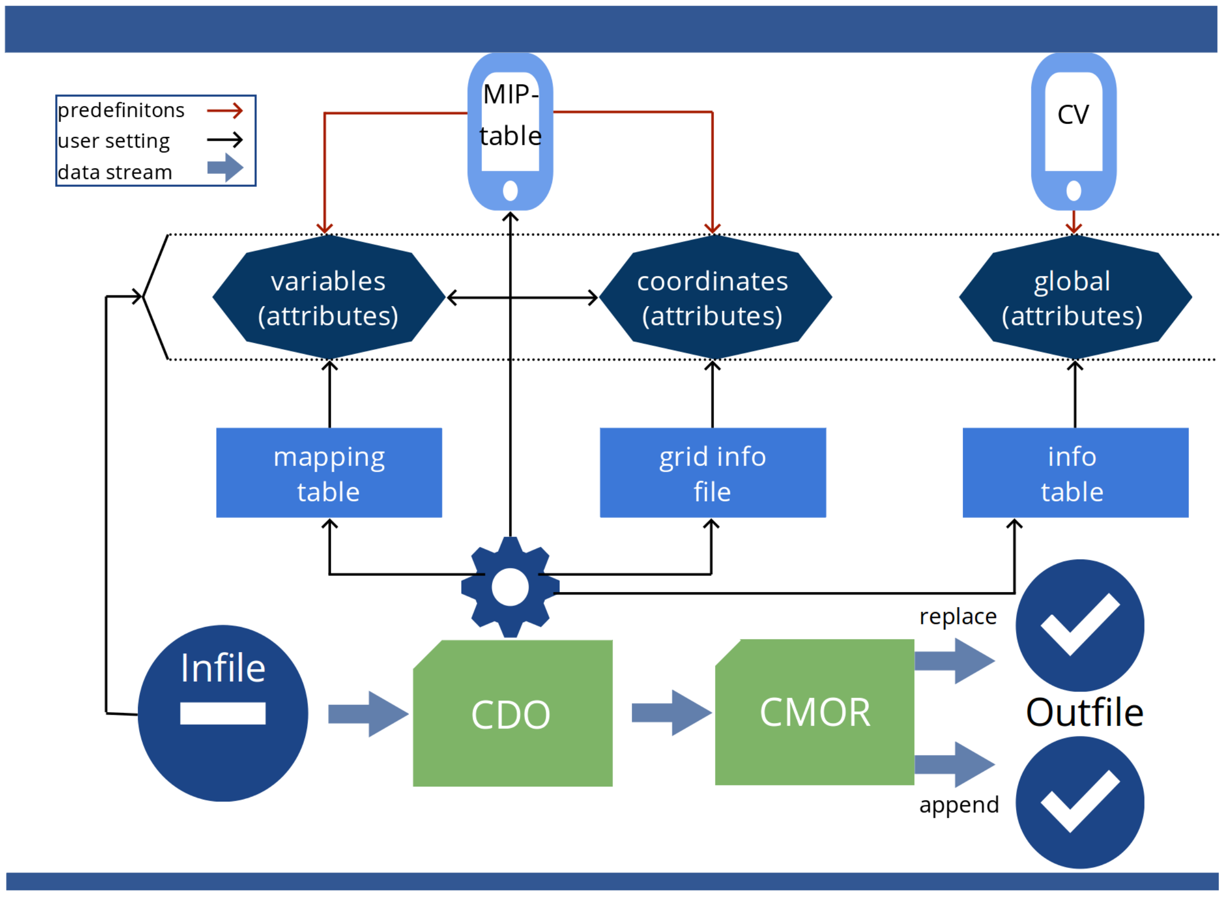

Figure 4.1.: Metadata workflow: Predefinitions and user setting options in **cdo cmor**

An order of precedence exist for the meta data configuration: Information given in *infile* may be overwritten by the corresponding information in the info table, mapping table or grid info file so that wrong *infile* information can be corrected. On top of that, **cdo cmor** features a command line 'mapping' (see section 4.2) enhancing the 'quick' and interactive application of the operator so that variable attributes and their coordinate information can be specified in the command line as well. Command line settings have the highest priority and overwrite all prior settings. The meta data and data stream is illustrated in figure 4.2.

However, usually *infile* does not contain the complete set of requested meta data. In this case, configuration files each linked to one of the three introduced meta data fields can be used to provide or correct the missing meta data: Info tables for global attributes (section 4.1), a mapping table for variables and their attributes (section 4.2.2) and a grid info file for coordinates and their attributes (section 4.3) (visualized in figure 4.1 by blue rectangles and the gear-wheel for the command line). These interfaces enable a comprehensive, reusable and easy adaptable configuration simplifying the use of the operator.

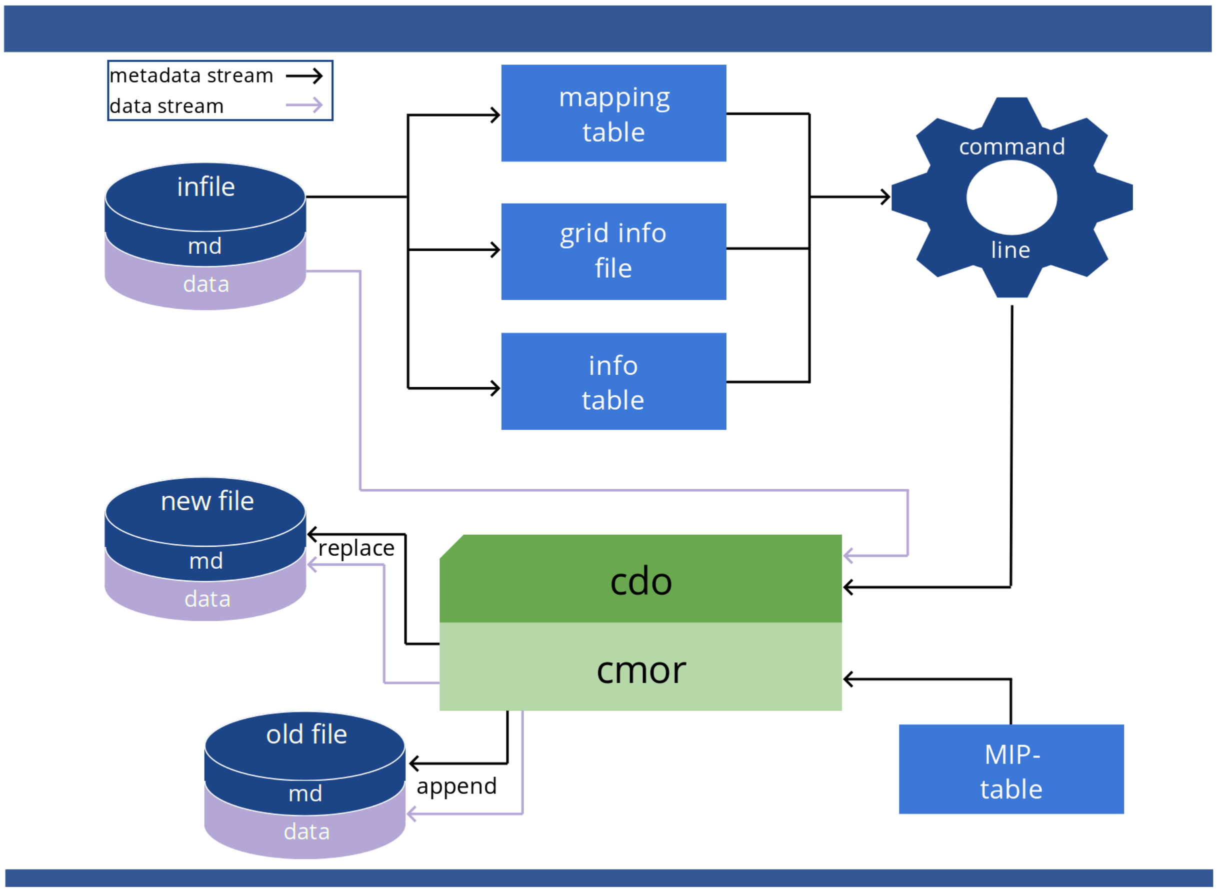

Figure 4.2.: Metadata and datas tream in **cdo cmor**. Wrong or missing information can be overwritten by meta data configuration files and command line parameters. All meta data information is collected, passed to CMOR and checked by CMOR with the help of the MIP-table.

The user can organize the meta data stream and specify meta data files via the command line. For each meta data table file, a keyword exists which can be given a value via the command line in the keyvalue list. In the following sections, the usage of the meta data files is explained. An info table (section 4.1) can also include control keywords next to global attributes (section 4.1.2). The mapping table enables linking model variables to CMOR variables reducing preprocessing efforts. Coordinate information can be provided via a grid info file (see section 4.3).

cdo cmor allows appending of output to an existing file. How to control this append mode is described in section 4.4. Methods of automatic background calculations in internal procedures of **cdo cmor** are introduced in section 4.5. That section is not essential for the correct usage of **cdo cmor** and may be skipped.

4.1. Info table

Info tables can be specified in a CSL with keyword `info`. If `info` is not used, a default file is searched by the operator in the current working directory (CWD). It is hidden and named '.cdocmorinfo'. An Info table contains a *keyvaluelist* with one `keyword`=*value* pair per line. All global attributes required by the project standard have to be given in Info tables. All specifications from files of that CSL are gathered to be one configuration for a single **cdo cmor** call.

In addition to global attributes, three groups of additional control keywords can be included: Filename keywords (`mip_table`, `mip_table_dir`, `mapping_table`, `mapping_table_dir`, `grid_info` and `grid_info_dir`), output control (`drs` and `drs_root`, see section 3.3), global attribute control and coordinate keywords. The operator can build complete file paths with the directory keywords. With the help of `project_id`, the operator can also build the full *MIP-table* name so that it can be sufficient to call 'cdo cmor,Amon,...'.

4.1.1. Examples

```
# The default info file '.cdocmorinfo' will be parsed.
# The following calls are equivalent:
    cdo cmor,cmip5_mip_tables/CMIP5_Amon,cn=tas example_interface.nc
    cdo cmor,cmip5_mip_tables/CMIP5_Amon,info='.cdocmorinfo',cn=tas example_interface.nc
# Since 'project_id' and 'mip_table_dir' are defined in '.cdocmorinfo', we can write
    cdo cmor,Amon,cn=tas example_interface.nc
```

4.1.2. Global attributes

CMOR version dependent

CMOR 3 (creates CMIP6 data standard)

As part of the CMIP standard, a set of global attributes is requested to be specified depending on the project. If `project_id` is not specified in an info table, the operator is able to detect the `project_id` by analyzing the *MIP-table* passed as argument 1 and continues to test for further global attributes depending on the project. It uses the first substring of the *MIP-table* file name separated by an underscore.

In CMIP6, this `project_id` is also used as `mip_era` and the other way around. All further *required attributes* depending on the project standard can be taken from the project's Control Vocabulary (CV). CMOR3 requires that this CV is expressed as a MIP-table named "<project_id>_CV.json" in the *MIP-table* directory. It contains the definitions of *required and recognized* attributes. Additionally, it includes allowed values and restrictions to the values of these attributes. Further information can be found in sections 4.1.2.1 and 4.1.2.2.

cdo cmor provides three keywords which simplify the specifications of a number of global attributes (their formats are given in the appendix B.1):

- `keep_all_attributes`: If its value is 'y', global attributes given in *infile* and in addition to the required global attributes will be kept and, if not overwritten by an info table specification, passed to CMOR. If its value is 'n', all global attributes given in *infile* will be discarded allthough they may be required by the standard.
- `member` combines the indices associated with realization, initialization, physics and forcing which describe the ensemble member.
- `parent_dates` contains information to create `parent_time_units`, `branch_time_in_parent`, `branch_time_in_child` which will be required if an experiment is branched from a parent experiment. The branch times are relative time values from a reference time defined by each experiment. It is assumed that the parent time units begin with "days since".

Attributes beginning with `parent` or `branch` are *required* if the current experiment is branched from another, called `parent_experiment`. Consider that *recognized* attributes can also have CV restrictions. Those who are unrestricted can be specified to supply the user with additional helpful information (e.g. `comment`, `history`, `references`, `variant_info`, summarized in appendix B.2). Unrestricted attributes will be transferred to the output without further checking if 'y' is specified for keyword `keep_all_attributes`.

We recommend distributing global attributes to different Info tables categorized by topics Model, Experiment and User (see Appendix) to keep a clear and reusable configuration structure. A single file including a whole set of attributes can be very long which complicates the regain or adaption of attributes. Furthermore, when converting output from the same model but for another experiment, only those attributes given in the Experiment table needs to be adapted.

CMOR 2 (creates CMIP5 and CORDEX data standard)

CMOR2 indirectly prescribes a subset of global attributes which are always recognized independent of the project's data standard (because these attributes correspond to arguments of the cmor_dataset function). They can be subdivided in global attributes which are always required (Table 4.1) and optional global attributes (Table 4.2).

Table 4.1.: Global attributes required by **cdo cmor** independently of the project if CDOs are installed with CMOR2

Keyword	**Example**	**Explanation**
`project_id`	CMIP5	Value must be the project which is found in the *MIP-table* passed as argument 1.
`experiment_id`	amip	
`model_id`	MPI-ESM-LR	
`source`	"MPI-ESM-LR 2011; URL: ht.... "	References for the model.
`instituie_id`	MPI-M	
`institution`	"Max Planck Institute for Meteorology"	
`contact`	cmip5-mpi-esm@dkrz.de	Contact for questions about the data.

Table 4.2.: Global attributes recognized by **cdo cmor** when CDOs are installed with CMOR2.

Keyword	**Example** (default format is string, otherwise mentioned)	**Explanation**
`realization`	1 (Integer)	Part of the ensemble member label.
`initialization_methods`	1 (Integer)	Part of the ensemble member label.
`physics_version`	1 (Integer)	Part of the ensemble member label.
`forcing`	oz	Description of forcing.
`leap_year`	2000 (Integer)	Can be specified if a specific year is skipped independent of the calendar
`leap_month`	11 (Integer)	Can be specified if a specific month is skipped independent of the calendar
`parent_experiment_id`	piControl	Similar to `experiment_id` but for parent
`parent_experiment_rip`	r1i1p1	Parent experiment ensemble member label.
`branch_time`	50 (Double)	Branch time in parent.

cdo cmor provides three keywords which simplify the specifications of a number of global attributes (their formats are given in the appendix B.1):

- `keep_all_attributes`: If its value is 'y', global attributes given in *infile* and in addition to the required global attributes will be kept and, if not overwritten by an info table specification, passed to CMOR. If its value is 'n', all global attributes given in *infile* will be discarded allthough they may be required by the standard.

- `member` combines the indices associated with realization, initialization and physics which describe the ensemble member.
- `parent_dates` contains information to create `parent_time_units` and `branch_time` which will be required if an experiment is branched from a parent experiment. The branch time is a relative time value from a reference time defined by the parent experiment. It is assumed that the parent time units begin with "days since".

.

As for CMOR3, we recommend to distribute global attributes to different Info tables categorized by topics Model, Experiment and User (see Appendix) to keep a clear and reusable structure.

4.1.2.1. CMIP5 and CORDEX compliant meta data standard

Attributes given in tables 4.1 and 4.2 are part of the project's meta data standard. Additional to these attributes, some more have to be passed to the operator in order to achieve CORDEX compliant format. Since the regional models used in CORDEX are driven by global ESM models, a domain and a driving model have to be specified within CORDEX meta data standard.

For both projects, an overview about all required global attributes is provided in appendix B.3 and B.4. Recent CORDEX projects (e.g. CORDEX-CORE) require that the tracking_id usually built by CMOR is constructed with a prefix in order to be compliant to the format of a PID. Therefore, the user can specify 'y' for the keyword `tracking_prefix` in order to use the project's prefix when building the tracking_id.

4.1.2.2. CMIP6 compliant meta data standard

All CMIP6 required attributes are shown in Table B.5. To simplify the correct configuration, the DKRZ provides a script at `c6dreq.dkrz.de/cdocmorinfo/index.html` which uses the CV file of CMIP6 to guarantee a CMIP6 compliant configuration set of global attributes. Background information to CMIP6 global attributes can be found in Taylor et al. (2019, `https://goo.gl/v1drZl`).

The DRS of CMIP6 provides a subdirectory <version>(see section 3.3). This is created by CMOR using the date the user calls the program. If the postprocessing takes longer than a day, a single variable may be saved in different version directories. This is a problem since a complete time series of a single variable of a single realization should be saved in one version directory. Therefore, cdo cmor offers the user to set a free string for this directory via keyword `version_date`.

4.2. Variable mapping

infiles may be available in a raw format containing variables without attributes, e.g. in GRIB-format where the variables are identified by codes. In this case, it is required to change each variable name to the correct `cmor_name` and provide requested attributes. The method to link *infile* variables to their corresponding **CMOR variable**s is called "mapping". In CMIP like projects, values for attributes `cmor_name`, `units`, `cell_methods` and `postive` are required. Note that the user's specifications of these attributes should describe the variable in *infile*. Additionally, it can be necessary to specify coordinate names. An overview of all valid mapping keywords is given in table 4.3.

cdo cmor provides two ways to map the *infile* variable: Via command line (section 4.2.1) and by using a mapping table (section 4.2.2). The command line method allows mapping of only one variable in each operator call. Note that the order of precedence (command line specifications overwrite table specifications which in turn overwrite *infile* specifications) enables the correction of prior settings.

For renaming an *infile* variable as its corresponding `cmor_name`, **cdo cmor** provides two additional *infile* variable selectors: `name` (which should be used if *infile* is formatted as netCDF) and `code` (which should be used if *infile* is formatted as GRIB). Only one value is allowed for both keywords. If one of these keywords is specified, a corresponding `cmor_name` must be specified as well.

The *infile* variable units can be provided as value of keyword `units`. `units` must be CMOR readable (UDUNITS conform) because CMOR will change it if the requested `units` are different. A `positive` attribute may be required if the variable repesents a vertical flux.

`cell_methods` denotes the aggregation method which was used to create the *infile* variable. A conversion of the variable is only possible if its `cell_methods` correspond to the value requested by the MIP-table. 'Mean' is used for temporal averages, 'point' for instantaneous, 'climate' for climatological mean of higher frequency data (e.g. the 30 year mean of monthly means), 'diurnal' for long-term means of one hour means covering a whole day and 'none' for static fields. Note that 'diurnal' is only valid in CMIP6.

Coordinates are also defined in the project standard and specific coordinate values, e.g. levels for vertical axes may be requested. Therefore, allowed coordinates are formulated as **CMOR variables**: For CMIP6, they are defined in the MIP-table 'CMIP6_coordinates.json' whereas for CMIP5, coordinates that are allowed for a specific variable have to be defined in the same MIP-table like the variable itself. The operator can distinguish axis types from *infile* (e.g. by interpreting the standard_name attribute of coordinate variables) and is therefore usually able to pass correct coordinate names to CMOR automatically.

One exception is the name of vertical pressure axes. The `cmor_name` of the axis describing **CMOR variable** needs to be provided as the value for keyword `z_axis`. Example: **CMOR variable** 'ta' in *MIP-table* "CMIP6_Amon.json" is requested on the vertical axis 'plev19'. In that case, `z_axis`=plev19 must be passed to the operator when producing 'ta' with "CMIP6_Amon.json". For scalar axes, the keyword `z_axis` needs to be specified as well if its value deviates from the requested one (see section 4.3.3). For configuring character coordinates with keyword `character_axis` please see section 4.3.4.

Table 4.3.: Target **CMOR variable** selection keyword (first line), *infile* variable selection keywords (second and third lines), variable attributes (following lines) and variable coordinates (last two lines) which can be specified. Note that only one *infile* variable selection keyword can be used, otherwise the operator gives an error.

Name	**Short name**	**Format and description**	**Default**
`cmor_name`	`cn`	The value must be in the *MIP-table* specified as argument 1. It is used to identify the target **CMOR variable**. If a mapping table is specified, it is also used to identify the appropriate table line (see 4.2.2).	Variable name in *infile* is interpreted as `cmor_name`.
`name`	`n`	Variable name in *infile* of the variable requested via `cmor_name`.	If neither 'name' nor 'code' is specified, no variable renaming is done.
`code`	`c`	Integer (Limit: Three digits). Value is the variable code in *infile* of the variable requested via `cmor_name`.	If neither 'name' nor 'code' is specified, no variable renaming is done.
`units`	`u`	String. Value must be readable by library 'udunits', e.g.: 'W m-2'. CMOR uses this library to convert *infile* data values if this is not the requested `units`.	If no value is registered, units are taken from *infile*. If no units information is available, the operator gives an error.
`positive`	`p`	Value is 'u' or 'd'. This attribute enables CMOR to switch the sign of a directed variable, e.g. radiation flux. 'u' stands for 'upward', 'd' for 'downward'.	No default.

`cell_methods`	cm	Value is one of 'm', 'p', 'c', 'd' and 'n' which stands for 'mean', 'point', 'climate', 'diurnal' and 'none' (see note in this section for more details). This is the aggregation method used for the chosen variable in *infile*. Its value is necessary to register the correct time axis.	The default is mean i.e. 'm'.
`variable_comment`	vc	String. Value gives additional information about the variable.	No default
`z_axis`	za	`cmor_name` of the coordinate. Optionally, values for the coordinate can be specified in a grid info file.	For a scalar z-axis: As in the *MIP-table*.
`character_axis`	ca	`cmor_name` name of the coordinate. Values for the coordinate need to be specified in a grid info file.	No default

4.2.1. Using the command line

The user can set all attributes displayed in Table 4.3 through the keyvalue list in the command line. These specifications overwrite all prior settings. A subset of variables can still be processed in one operator call, however, it can be problematic since the specified variable attributes are assigned to all *infile* variables. Therefore, it is recommended to specify a `cmor_name` when variable attributes are specified. If renaming is used as well, only one variable can be processed in each operator call. These cases are illustrated in the following examples.

```
# Subset of variables are processed (Innames = Infile variable names):
    cdo cmor,MIP-table infile
                                                              #CMOR names are innames*
    cdo cmor,MIP-table,cmor_name=cnames infile              #select by cnames
                                                              #CMOR names are cnames*
    # units 'K' is assigned to all CMOR variables listed in cnames:
    cdo cmor,MIP-table,cmor_name=cnames,units=K infile      #select by cnames
                                                              #CMOR names are cnames*


# Active renaming; only one variable can be processed:
    cdo cmor,MIP-table,name=inname,cmor_name=cname infile    #select by inname
                                                              #CMOR name is cname*
    cdo cmor,MIP-table,code=incode,cmor_name=cname infile    #select by incode
                                                              #CMOR name is cname*
```

*In few cases, the cmor_name is not equal to the name of the variable in outfile, e.g. for cn=co2clim where the variable name is co2 in the outfile.

4.2.2. Using a mapping table file

cdo cmor offers a way to rename a subset of variables and provide them with individual attributes in the same operator call which is beneficial e.g. in the operational application. This is realized by a mapping table file which can be registered via keyword `mapping_table` and which is formatted like a fortran namelist. Each line of this file is related to one **CMOR variable** and contains the variable's attributes. An example of three lines of a mapping table is given in the following.

```
&parameter project_mip_table=Amon cn=tas       n=temp2    c=167  p=" " u="K"  cm="m" /
&parameter project_mip_table=Amon cn=sos       n=sss      c=016  u="psu"      cm=m /
&parameter project_mip_table=Amon cn=msftbarot n=psitro   c=027  u="kg s-1"   cm=m /
```

Consider format constraints when preparing a mapping table:

1. Each line must begin with the entry keyword '&parameter' and must end with '/'.
2. If a value contains blanks, it must be put in quotation marks.
3. In contrast to the interactive mapping, one can also denote both, name and code in the table.
4. Both, the order of further keywords in each line and additional blanks between value and next keyword do not matter.

In order to link each mapping table line to one specific **CMOR variable**, the keyword `project_mip_table` exists which is the substring of the MIP-table name after '<project_id>_', for example: 'Amon'. Both keywords `cmor_name` and `project_mip_table` should be specified in each mapping table line for a clear identification of the matching **CMOR variable**.

The mapping procedure can be subdivided into the mapping table line (MTL) selection and *infile* variable (IV) selection. The MTL selection depends on whether `cmor_name` is specified in the command line or not. Subsequently, the IV is selected with the help of a keyword from the selected MTL. Both procedures are explained by the implemented chronological processing sequence in the following.

MTL selection if `cmor_name` is specified in the command line (recommended approach)

In this case, the target **CMOR variable** is determined in the operator call with the help of `cmor_name` and the *MIP-table* (argument 1). The mapping table line which contains the `cmor_name` that matches the one specified in the command line is identified. If no line with a corresponding `cmor_name` is found, the mapping table is not processed for the `cmor_name` specified in the command line. Otherwise, the identified line is tested for keyword `project_mip_table`:

- In case the identified line contains a `project_mip_table` keyvalue:

  It is tested whether it matches the *MIP-table* passed as argument 1. There can be multiple lines in the mapping table with the same `cmor_name` because, for different `project_mip_tables`, the requested variable and also the corresponding provided *infile* variable can be different. If the `project_mip_table` keyvalue disagrees with the *MIP-table*, this line is rejected and the next lines are tested similar to the procedure introduced before. If the `project_mip_table` value matches the *MIP-table* passed as argument 1, this line is taken for mapping.

- In case the identified line does not contain a `project_mip_table` keyvalue:

  The identified line is saved but next lines are also tested. The last line of matching `cmor_names` is taken for mapping.

Selection of the *infile* variable:

As next step, the corresponding *infile* variable needs to be detected. Therefore, variable selectors `name` and `code` are identified from that line and compared with *infile* variables. Keyword `name` is applied in case the *infile* has NetCDF format and `code` in case the *infile* has GRIB format. If a variable in *infile* has the corresponding name, all further attributes of this line are assigned to this variable. In case `name` and `code` are missing, `cmor_name` is also used as *infile* variable selector. If no selector attribute corresponds to an *infile* variable, the line in the mapping table will be ignored and next lines are tested.

MTL and *infile* variable selection if `cmor_name` is not specified in the command line:

As usual, all variables in *infile* will be processed. However, the *infile* variable `names` are not considered as `cmor_names` but are used as MTL selection keyword (*infile* variable `code` for GRIB *infiles*). That is, if a mapping table line contains a variable selector keyword which matches an *infile* variable, this line is used for mapping. Again, the mapping table line should contain a *project_mip_table* keyword which is checked against the *MIP-table* given as argument 1 in the way described before.

Overwriting mapping table configurations with command line mapping:

If, in addition to a mapping table, a variable selection key (`name` or `code`) and `cmor_name` are specified in the command line, the IV selection changes: The value of keyword `name` found in the mapping table line is ignored and the variable from *infile* is chosen by the `name` specified in the command line. Furthermore, attributes configured in the command line overwrites attributes configured via mapping table. Therefore, the combination of both mapping methods can be reasonable e.g. if the origin model variable changed after the mapping table was constructed.

4.2.3. Examples

1. Simple example of preparation and application of a mapping table:

Szenario: Your model produces output in GRIB-format where variables are identified via `code`. You are interested in the MIP-compliant conversion of near-surface temperature tas (code 167).

For creating a one-line mapping table ”mapping_table.txt”, execute:

```
  echo '&parameter cn=tas c=167 u=K cm=m /' >>mapping_table.txt
```

With this line, the following two **cdo cmor** calls are equivalent:

```
  cdo cmor,Amon,cn=tas,c=167,u=K,cm=m example_mapping.grb
  cdo cmor,Amon,mt=mapping_table.txt example_mapping.grb
```

2. Use the same mapping table line but applied on an *infile* variable with another name:

Szenario: Due to model development, model variables are identified with names now but you do not want to adapt the mapping table.

```
  cdo cmor,Amon,cn=tas,n=T_2M,mt=mapping_table.txt   example_T_2M.nc
```

3. Use the same mapping table line but overwrite values:

Szenario: You would like to use the mapping table from someone but you want also change an attribute.

```
  cdo cmor,Amon,cn=tas,n=T_2M,mt=mapping_table.txt,u=degC example_T_2M_celsius.nc
```

4. Use `project_mip_table`:

Szenario: If a variable, e.g. with `cmor_name` tas, comes from the atmosphere model for table Amon, but from the ocean model for table day.

```
# we have to remove the old mapping table
# since we now add project_mip_table:
  rm mapping_table.txt

  echo '&parameter cn=tas n=atmosTas u=K cm=m project_mip_table=Amon/' >>mapping_table.txt
  echo '&parameter cn=tas n=oceanTas u=K cm=m project_mip_table=day/' >>mapping_table.txt
  cdo cmor,Amon,cn=tas,mt=mapping_table.txt example_temp_mon.nc # Uses atmosTas
  cdo cmor,day,cn=tas,mt=mapping_table.txt example_temp_day.nc # Uses oceanTas
```

Note that you have to specify a `cmor_name` because otherwise, **cdo cmor** processes all *infile* variables and maps via name.

5. Use mapping table for a subset of variables.

Szenario: Assume, you finally have the ”perfect” mapping table ”mtPERFECT.txt” where all model output variables are assigned to the corresponding `cmor_name`.

```
  cdo cmor,Amon,mt=mtPERFECT.txt example_collect.grb
```

The last operation produces several files, where `name` is used as entry keyword for the mapping table.

4.3. Coordinates

Coordinates are also provided as CMOR variables in the CMIP standard (for CMIP6 in "CMIP6_coordinates.json", for CMIP5 in each MIP_table). Target variables can be requested on a specific coordinate variable (e.g. tas on height2m). This is not always the case: There are variables requested on dimension "alev" which means atmospheric model level - here various coordinates are valid. Therefore, CMOR requires the `cmor_name` of the axis a variable is submitted on as input.

The interface of the CDOs can retrieve much temporal and spatial information from *infile*. Attributes 'standard_name' and 'units' of coordinate variables are read and used to define axis types according to the CF-conventions, or, in case of an *infile* in GRIB format, accoding to GRIB tables. The output of **cdo sinfo**, **cdo griddes**, **cdo zaxisdes** and **cdo showtimestamp** show how CDO handles coordinates of *infile*. The information is used to pass the correct axis names to CMOR.

However, sometimes this retrieved information can be insufficient for the conversion to CMIP standard. In the following, it is discussed how to pass additional coordinates information to the operator. One option is to provide a grid info file (section 4.3.1). The time axis (section 4.3.2) and scalar and character axes (section 4.3.3 and 4.3.4) have specific requirements and the operator therefore offers additional configuration options discussed in the corresponding sections. All required grid information for all target variables needs to be available during one operator call.

4.3.1. Grid info file

A grid info file specified via `grid_info` may be either in a table format equivalent to the Info table or in netCDF format. The table format grid info file is intended for (re-)defining coordinates which are assigned in the mapping to particular axes of target variables. The processing of a grid info file in table format is equal to an Info table file however keywords can only be either coordinate names or time keywords - corresponding use cases are given in sections 4.3.2, 4.3.3 and 4.3.4. Not all axis definitions included in a grid Info table file are completely applied but only those which are pointed at by mapping keywords.

A more powerful grid info file is of netCDF format. This type has the purpose to completely substitute the *infile* grid and can also be used to (re-)define coordinates. Thus, it contains all required coordinate variables and their attributes as well as their bounds. Note that the z-axis will be switched too unless the user specifies the keyword `switch_z`. In contrast to a grid info file in table format, all information in a netCDF grid info file is used whenever it is specified and the operator aborts, if the substituted grid does not fit to the *infile* variable. Such a grid info netCDF file can be for example a compliant output file from an earlier conversion.

The operator can only retrieve a grid from a netCDF grid info file if a variable exist in that file which points to the the correct coordinate descriptions. Only one grid can be passed to the operator and it will be the grid information of the first variable the operator finds in the netCDF grid info file. The operator checks whether the dimension size of the axis in *infile* which is to be substituted fits to the total numbers of values specified for an axis provided in a grid info file and stops if not.

4.3.1.1. Examples

Grid info table

In the operator call:

```
cdo cmor,Amon,za=height2m,cn=tas,gi=gitable.txt example_interface.nc
```

a keyvalue "z_axis=height2m" is specified. Now, it is searched for information about the height2m axis in the grid info file "gitable.txt" (For more information about height2m see section 4.3.3). If no reference for height2m is given during mapping, information about height2m in the grid info file is ignored.

Grid info netCDF file

Models participating in project CORDEX usually use grids with grid mapping parameter describing the relation between the used coordinates and the true latitude and longitude coordintaes. However, model output raw data does not contain required grid information like grid cell boundaries or mapping attributes. If project compliant grid data is available, you can substitute the incomplete grid information of the raw data:

```
cdo cmor,mon,i=cdocmorinfoCORDEX,gi=example_gridinfo_CCLM4-8-17.nc,cn=tas example_CCLM.nc
```

4.3.2. Time axis

CMOR distinguishes five time axes : time, time1, time2, time3 and none for fixed fields. The user can detect a CMOR variable's time axis by looking at its dimension names. Each of these time axes correspond to a specific average method or time cell method which describes how a CMOR variable is aggregated. The user needs to provide the cell_methods attribute for each variable so that the operator can check whether this method is compliant to the requested time axis.

Technically, each time axis has different requirements on the time axis and time bounds. If the requested time cell_methods (section 4.2) for a target variable is neither 'none' nor 'point', the standardized output includes time values and time bounds. In case the cell_methods is 'point', the exact time values for when the data is valid are required. 'none' means that the target variable has no time axis and its values are fixed.

The input requested by CMOR for time axes time, time2 and time3 are the time bounds whereas for time1 it is the time values. If the *infile* includes time bounds, they are passed to CMOR. Otherwise, the operator can calculate time bounds on the basis of time values (section 4.5) and with the help of keywords. In case the time axis is time1 the provided time values must be exact. The *infile* time bounds can be discarded by specifying 'y' for keyword `time_bounds` which causes the operator to ignore *infile* time bounds and to use the calculation method based on time values. An overview of the different time axes are given in Table 4.4.

Table 4.4.: Link of time axis with requested `cell_methods`, format of time values and time bounds as well as required user input.

Time axis	`cell_methods`	time values	time bounds	required user input
none	none			
time	mean	At the midpoint of average intervall given by time bounds. CMOR determines it automatically.	The interval must cover the timestep in the frequency the variable is requested for.	Either the correct time bounds. Or time values: time bounds are created with the frequency the variable is requested on.
time1	point	When variable is written.		The exact time values.
time2	climate		Climatology time bounds will cover a month and years	Either climatology time bounds. Or time values with the exact month the variable is valid for and attribute `climatology_interval` (see text).
time3	diurnal		Time bounds cover an hour and a month	Either time bounds. Or time values indicating the exact month and hour the values are valid for.

The standardized file includes a relative time axis where values are the temporal distance to a reference time (usually in days). The format of time values and time bounds in *infile* however can be either relative or absolute which includes complete date and time information for every record. In case the input variable uses an absolute axis, the operator is able to transforms it to a relative time axis with the keyword `required_time_units`. Five different recommendations how to set this time units can be taken from Taylor et al. 2019, https://goo.gl/neswPr. Note that time related keywords discussed in this section can be either specified in Info tables or in a grid Info table file.

Furthermore, the calendar is mandatory (normally given in *infile*) and can be specified via keyword `calendar`. Its default value is 'standard'.

If a climatological time axis is requested, the first year and last year of the averaging interval must be provided via keyword `climatology_interval`. This is necessary because the averaging period can not be deduced from the *infile* information. All keywords referring to temporal information are listed in Table 4.5.

Table 4.5.: Time related keywords.

Name	**Short name**	**Format and description**	**Default**
`required_time_units` (also settable in command line and *infile*)	`rtu`	'<Frequency>since <Year>-<Month>- <Day><Hours>: <Minutes>: <Seconds>', for example: 'days since 1979-1-1 00:00:00'.	No default
`calendar`		Value is one of 'standard' ('gregorian'), 'proleptic_gregorian', '360_day', 'noleap' and 'all_leap'. The calendar depends on the model configuration.	'standard'
`climatology_interval`		Two comma separated years as integers. E.g.: 2001,2010	No default
`time_bounds`		'c' for calculating time_bounds from *infile* time values and overwriting *infile* time_bounds.	If available, using provided time_bounds from *infile*. Otherwise: 'c'.

4.3.3. Scalar axes

If a dimension has size one, the corresponding axis is referred to as "scalar" and has a specific requested format. E.g., if a variable is requested for only one specific level, the project standard provides that this level is not a dimension but a coordinate variable.

Although a specific coordinate value of a scalar axis is requested by the project standard, the value may deviate if a valid range is defined. E.g., for the scalar axis 'height2m', values between 1.0 and 10.0 are also allowed. The operator uses the default value for the scalar axis which can be determined from the its name (e.g. 2m for height2m) without further specifications. If however the level used in *infile* (i.e. the model) is different from the requested one, the user has two options: Either to provide a grid info netCDF file with a standard compliant scalar axis variable containing the deviating value or to use an info table where

- the <z_axis_name>(`cmor_name` of the corresponding scalar axis **CMOR variable**) needs to be provided during mapping as value of keyword `z_axis` (see section 4.2), e.g.: `z_axis`=basin.
- the level used in *infile* needs to be specified in the grid info file for a keyword named like the z_axis in the format: <z_axis_name>=<level_value>, e.g.: height2m=1.5

4.3.3.1. Examples

The **CMOR variable** with `cmor_name` 'tas' in MIP-table 'Amon' (Near-surface Air Temperature) is requested with scalar coordinate 'height2m' with '2m' as default value. If the model outputs the corresponding tas at 1.5m, the default of 2m can be overwritten by:

```
# in the mapping, e.g. command line mapping:
  cdo cmor,Amon,cn=tas,z_axis=height2m example_interface.nc

# in grid Info table file:
  height2m=1.5
```

4.3.4. Labeled axes

"When data is representative of geographic regions or area types which can be identified by names but which have complex boundaries that cannot practically be specified using longitude and latitude boundary coordinates, a labeled axis should be used to identify the regions. Character strings labelling the elements of an axis are regarded as string-valued auxiliary coordinate variables" (CF 6.1 and 6.1.1). Accordingly, two standard names are available for those coordinate variables (referred to as character axis): 'region' and 'area_type'.

Such a character coordinate variable is usually not provided in *infile* - often only dimension indices are given in *infile*. For that case, the operator offers a configuration similar to the scalar axis configuration: The name of this axis as well as its coordinate names can be registered in an Info table. The order of character axis values must correspond to the dimension indices sequence.

- The <character_axis_name>(`cmor_name` of the corresponding character axis **CMOR variable**) can be configured with the keyword: `character_axis` (see section 4.2), e.g.: `ca`=basin.
- A connected second keyword in the grid info table then defines the values of the axis: <character_axis_name>=<coordinate_names>, e.g.: basin=global_ocean,atlantic_arctic_ocean,indian_pacific_ocean

Each elements for a labeled axis can be provided as individual variables. To cause the operator to merge these elements together into one variable, they have to be specified as values for `name` respective `code` in the mapping table line of the corresponding `cmor_name`. The operator checks if the field sizes of all individual labeled elements agree and converts the dimension which contains only one value into a character axis.

4.3.4.1. Examples

(no files yet in the example directory)

Variable msftyz in CMIP6_Omon.json is requested with coordinates latitude, rho, basin and time. "basin" is requested with elements "atlantic_arctic_ocean", "indian_pacific_ocean" and "global_ocean".

Szenario: A basin describing string-valued coordinate variable is not given in *infile*.

```
# Provide the string-valued coordinate variable name during mapping,
# e.g. command line mapping:
  cdo cmor,Omon,cn=msftyz,character_axis=basin infile

# Specify the coordinate variable name and its values
# in grid info table file
# in the order they are provided in the basin dimension:
  basin=global_ocean,atlantic_arctic_ocean,indian_pacific_ocean
```

Szenario: Additionally to the first scenario, instead of msftyz(lat,rho,basin,time) the basin coordinates are given as variables atlantic_var(lat,rho,1,time), indian_var(lat,rho,1, time) and global_var(lat,rho,1,time).

```
# Cause the operator to merge variables to one variable
&parameter cn=msftyz n=atlantic_arctic_var,indian_var,global_var
```

4.3.5. Vertical parametric axes

The CMIP standard allows for vertical parametric axes if a variable is requested on 'alev's (atmospheric model levels). These axes are built on the basis of a formula which contains parameters. E.g., the hybrid sigma pressure coordinate named 'alternate_hybrid_sigma' is built by the formula:

$$p(n, lev, j, i) = a_p(lev) + b(lev) * p_s(n, j, i)$$

where $p(n, lev, j, i)$ is the pressure at gridpoint (n, lev, j, i), $a_p(lev)$ and $b(lev)$ are components of the hybrid coordinate at level lev and $p_s(n, j, i)$ is the surface pressure at horizontal gridpoint (j, i) and time (n). [1] In standard output, variables ap(lev), b(lev), ps(n,j,i), ap_bnds(lev,bnds) and b_bnds(lev,bnds) are available in addition to the variable lev(lev) and lev_bnds(lev, bnds). These parameters have to be passed to the operator.

- If *infile* is a netCDF file:

  The recommended procedure is to provide the coordinate variables according to the CMIP standard and CF-conventions: The vertical coordinate variable (e.g. named lev but of hybrid sigma pressure type) is associated with its definition by the value ("atmosphere_hybrid_sigma_pressure_coordinate") of the `standard_name` attribute (Usually, the definition formula is also provided in a `formula` attribute). The terms in the definition formula ($a_p, b * p_s$) are associated with the file variables (ap, b, ps) by the `formula_terms` attribute. As an alternative, variables named 'hyai' and b 'hybi' will be automatically associated with ap and b (The processing with output of atmospheric model echam).

  The `formula_terms` attribute takes a string value of the form " term: variable" i.e. "ap: ap b: b ps: ps". The order of elements does not matter. (See CF-conventions Appendix D).

- In a grib formatted *infile*:

  The operator identifies a hybrid alternate axis if the key 'indicatorOfTypeOfLevel' (octet 10) has either the value 109 or 110. The corresponding parameter ap and b then should be saved for gribapi key "pv".

For variables with this axis type, the surface pressure is often saved separately, however, for the operator a surface pressure variable has to be available in *infile* or in a substitution grid info netCDF file. The operator identifies the surface pressure via variable name 'ps'. If a `mapping_table` is specified, the `name` which corresponds to `cmor_name`=ps is used to find the correct *infile* variable. If ps is not found but 'alternate_hybrid_sigma' is requested, the operator gives an error.

4.4. Output control

The operator offers two keywords to modify the path construction to the outfile: `drs` and `drs_root`. The value of `drs` determines whether a directory structure with DRS components according to the project's build-rule is built (see section 3.3) and the value of `drs_root` defines where this structure is rooted.

Standardized data is compressed per default with deflate_level 1 and *shuffle*. The user can set attribte `deflate_level` to -1 in order to create uncompressed data. If `deflate_level` is set to 0, the output will be compressed with shuffle only.

cdo cmor allows the user to append data to a 'chunk' file which is an incomplete CMOR output file which contains only data for a temporal part of a period. How to control this append mode is discussed in the following.

4.4.1. Append mode

The operator replaces files if the `output_mode` is set to 'r' (REPLACE) or append data to a chunk if the `output_mode` is set to 'a (APPEND). The default output mode of **cdo cmor** is 'APPEND'. **cdo cmor**

[1] Right now, only this 'alternate_hybrid_sigma' axis is enabled and test data for other axes built by a formula is required to guarantee a correct processing in CDOs.

tries to open a chunk which can either be specified via `last_chunk` or read from a chunk description file. On success, data will be appended.

The chunk description file is created every time cdo cmor is applied in append mode. This is because, if the append mode is consecutively used several times, e.g. in operational usage, the file name continuously changes which makes the user specification of `last_chunk` difficult. Therefore, the name of the successfully created outfile is written into a file of the form: '.CHUNK_FILE_ <`cmor_name`>_ <miptab_freq>_ <model_id>_ <experiment_id>_ <member>.txt' and saves it in the working directory. E.g.: 'CHUNK_FILE_ tas_ mon_ MPI-ESM_ Amon_ r1i1p1.txt'. When **cdo cmor** is started in append mode and with enabled path creation `drs`=y again, the file name saved in 'CHUNK_FILE_ tas_ mon_ MPI-ESM_ Amon_ r1i1p1.txt' is searched. If no valid chunk is found, the output mode switches to REPLACE.

In the process of appending, it is checked that the temporal interval between the previous chunk and the new data leaves no gap. The maximal file size including chunk size and new data size can be specified for keyword `max_size`. It has the default value 2Gb. If the size is exceeded, the operator switches to REPLACE mode. Table 4.6 gives an overview about all output control keywords.

Table 4.6.: Keywords to control the output of **cdo cmor**

Name	**Short name**	**Format and description**	**Default**
`output_mode`	`om`	Character. Value is either 'r' for CMOR replace mode or 'a' for CMOR append mode. In replace mode, CMOR produces a new file.	'r' for replace
`last_chunk`	`lc`	CSL. Values are chunk file names corresponding to the order of requested variables via 'cmor_name'.	For CMIP5 and 'drs'=y, file names are built with attributes in the form: 'CHUNK_FILE_ <cmor_name>_ <miptab_freq>_ <model_id>_ <experiment_id>_ <member>.txt' from where the operator reads each chunk file name.
`max_size`	`ms`	Integer. Value is the upper limit for the size of an output file in gigabyte and only used in append mode.	2
`drs`	`d`	Character. Value is either 'y' or 'n'. 'y': Directory structure according to a template (see section 3.3) is built. 'n': No directory structure is built and the file is saved in the directory specified via `drs_root`.	y. Note that, for ESGF publication, the data have to be submitted with the DRS structure.
`drs_root`	`dr`	String. Value is the path where the directory structure (DRS) is rooted.	Current working directory
`deflate_level`	`dl`	Integer. Value is level on which the varible is compressed.	1

4.4.2. Examples

Szenario: Your postprocessing is executed whenever one model simulation year output has been written. However, the final file should contain more than one year. The file you produced earlier when doing the example in section 3.1.1 is your chunk and data for year 2002 should be appended.

```
DRSpath=CMIP5/output/MPI-M/MPI-ESM-LR/amip/mon/atmos/tas/r1i1p1/
cdo cmor,Amon,om=a,mt=mapping_table.txt,
lc=${DRSpath}tas_Amon_MPI-ESM-LR_amip_r1i1p1_200101-200112.nc example_temp2002.nc
```

During the processing the operator creates the file 'CHUNK_FILE_tas_Amon_MPI-ESM_amip_r1i1p1.txt' where the name of the last chunk is saved. If you want to add new data to the chunk, you do not have to specify the `last_chunk` by yourself.

```
cdo cmor,Amon,om=a,mt=mapping_table.txt example_temp2003.nc
```

4.5. Internal procedures

This chapter illuminates background operator functions that automatically help during the conversion process.

4.5.1. Time coordinate

If `cell_methods` is not 'none', what is actually passed to CMOR are time bounds instead of time values. The operator, however, can calculate these bounds based on provided time values and a frequency which is usually deduced from the *MIP-table* name. Since CMOR expects boundaries to leave no gap, the processed time bounds always cover the whole unit of the frequency: If the frequency is yearly, a time value is valid for the whole year. E.g., if a time value is 01-23-1954 00:00:00 and frequency is yearly, the lower bound is 01-01-1954 00:00:00 and the upper bound is 01-01-1955 00:00:00. The original time value is moved to the mid point of this range by CMOR, which would be 07-01-1954 12:00:00 in the example if a 'standard' calendar is used. For requests with subdaily frequencies, this method is deviated: The upper bound t_+ and lower bound t_- are calculated by $t_\pm = t \pm 1/2 * f$ where t is the time value and f is the frequency. Here it is even more important that the provided time value is definitely correct.

If no frequency can be deduced from the *MIP-table* name, **cdo cmor** tries to derive it from *infile* by counting time steps and determine the covered temporal range by these. However, if the operator is part of a CDO chain this derivation is not possible because *infile* must be opened a second time. This is due to the fact that CDO intends to read one record of all variables per timestep which cannot be part of the 'get_frequency' function.

The frequency derivation from *infile* relies on two steps: First, $\frac{N_t}{\Delta yr}$ is calculated where N_t is the number of time steps in *infile* and Δyr the covered temporal range in years. If this term is 1, frequency is annual, if it is 12, frequency is monthly, and so on. If no clear frequency assignment is possible, as a next step $\frac{N_{t1}}{\Delta mon}$ is calculated which is the number of time steps in the first year N_{t1} divided by the covered months in the first year. If $\frac{N_{t1}}{\Delta mon} > 31 * 8$, **cdo cmor** gives an error because a sub-3hourly frequency is not yet enabled. If $\frac{N_{t1}}{\Delta mon} > 31 * 4$, frequency is 3hourly, if $\frac{N_{t1}}{\Delta mon} > 31$, frequency is 6hourly and so on up to monthly.

4.5.2. Spatial boundaries

CMOR requires spatial boundaries for axes which define the valid area respective covered height of a grid cell. If boundaries for a spatial axis are not implemented in *infile*, **cdo cmor** will interpolate the given grid point coordinates to the mid point between adjacent grid points, respectively, in order to build boundaries.

In case of a regular grid, the northern and southern boundaries latitudes are constant along the northern and southern boundaries, repsectively, as well as the western and eastern boundary longitudes are constant along the western and eastern boundaries, respectively. Four values are required by CMOR for each cardinal direction at each grid point. If needed, **cdo cmor** calculates them by averaging the longitutes as well as latitudes of two adjacent grid points, respectively. The upper and lower boundaries for z-axis levels are also calculated by averaging two adjacent level values, respectively.

In case of a curvilinear grid, CMOR requires grid cell corners and for all four grid cell corners, both, longitues and latitudes must be specified. If needed, **cdo cmor** calculates them with a piecewise bilinear interpolation: Half-longitudes (-latitudes) are calculated by averaging adjacent longitudes (latitudes). The grid cell corner longitudes (latidues) can be derived by averaging adjacent half-longitudes (-latitudes). I.e., the grid cell corner longitudes (latitudes) are the half-longitudes-on-half-latidues (half-latitudes-on-half-longitudes). If the absolute difference between two adjacent longitude values is bigger than 180 degrees, 180 degrees are added (substracted) to the average of these values if it is lower (higher) than 180 degrees. This procedure is observed to produce plausible bounds at the transmission from 360 to 0 degrees on the grid and prohibits values lower 0 or higher 360 degrees. Since this is an expensive and possibly imprecise approach it is highly recommended to deliver the bounds.

A. Table of all keywords except global attributes

Abbrev. in tables: Column **Necessary Information:** Y = yes: from key, from Infile or from default; (Y) = yes if request demands it; N = No). Column **Associated with topic:** E: Experiment, M: Model, U: User, G: Grid.

Function	**Key name** *Italic*: Also specifiable in info files.	**Key short name**	**Necessary information**	**Format**	**Default** if not available from key or infile
Variable Selector	cmor_name	cn	Y	Must be in MIP-table	-
Variable Selector	name	n	N	-	-
Variable Selector	code	c	N	Three digits integer. Equivalent to GRIB-code	-
Metadata File	info	i	Y	CSL of Filenames	'.cdomorinfo' in the current working directory
Metadata File	*grid_info*	gi	Y	Filename. Grid dimensions must fit to infile variable dimensions.	-
Metadata File	*grid_info_dir*	-	N	Filepath	-
Metadata File	*mapping_table*	mt	N	Filename	-
Metadata File	*mapping_table_dir*	-	N	Filepath	-
Metadata File	*mip_table_dir*	-	N	Filepath	-
Coordinate Info	*required_time_units*	rtu	Y	String: '<Frequency> since <Year>- <Month>- <Day><Hours>: <Minutes>: <Seconds>'	-
Coordinate Info	*calendar*	-	Y	5 options	-
Coordinate Info	*climatology_interval*	-	(Y)	Two comma separated years (integer).	-
Coordinate Info	*tbnds_ign*	-	Y	Character. y for ignoring infile time bounds variable, n for not.	n
CMOR variable selector in mapping table	project_mip_table	pmt	N	Substring of MIP-table filename after '_'. Only assignable in a mapping table.	-
Variable Mapping	units	u	Y	String. Must be readable from UD_units library	-
Variable Mapping	cell_methods	cm	Y	5 options	mean
Variable Mapping	positive	p	(Y)	Character. d for downward, u for upward	-
Variable Mapping	variable_comment	vc	N	String	-
Variable Mapping	character_axis	ca	N	Cmor axis label	-

Variable Mapping	z_axis	za	N	Cmor axis label	-
Output Control	output_mode	om	Y	Character. r for replace or a for append	r
Output Control	max_size	ms	Y	Integer. Unit: Gb	2
Output Control	last_chunk	lc	N	CSL of Filenames	-
Output Control	drs	d	Y	Character. 'y' for building path and 'n' for not	'y'
Output Control	drs_root	dr	Y	Path where DRS path is created	current working directory

B. Tables of global attributes

B.1. Keywords to simplify global attribute configuration

Keyword	Associated global attributes in CMIP6	Associated global attributes in CMIP5	Format
`keep_all_attributes`	all	all	'y' (default) or 'n'
`parent_dates`	`parent_time_units`, `branch_time_in_parent`, `branch_time_in_child`	`branch_time`	Two comma separated integers: <parentReferenceDate>, <parentBranchDate>. where each Date has the format: <year><month ><day>. E.g.: 18500101,18500201.
`member`	`realization_index`, `initialization_index`, `physics_index`, `forcing_index`	`realization`, `initialization_method`, `physics_version`	r%di%dp%d (CMIP5) or r%di%dp%df%d (CMIP6) where %d is each index in integer format.

B.2. Unrestricted global attributes

Keyword	Value example	Explanation
`comment`	any	Additional information.
`history`	any	Additional information.
`references`	ECHAM6: Stevens, ...	Citations.
`variant_info`	"GHG, Oz, ..."	Characteristics of ensemble member.

B.3. Attributes recognized or required by the CMIP5 standard

Global attribute.	**Associated with topic**	**Example** Default type is String, otherwise mentioned.	**Necessary information**	Explanation
`project_id`	E	CMIP5	Y	Value must be equivalent to the project which can be found in the *MIP-table* passed as argument 1.
`experiment_id`	E	amip	Y	
`realization`	E	1 (Integer)	Y	Part of the ensemble member label.
`initialization_method`	E	1 (Integer)	Y	Part of the ensemble member label.
`physics_version`	E	1 (Integer)	Y	Part of the ensemble member label.
`forcing`	E	oz	N	Description of forcing.

`comment`	E		N	
`parent_experiment_id`	E	piControl	(Y)	Similar to `experiment_id` but for parent
`parent_experiment_rip`	E	r1i1p1	(Y)	Parent experiment ensemble member label.
`branch_time`	E	50 (Double)	(Y)	Branch time in parent.
`model_id`	M	MPI-ESM	Y	
`source`	M	"MPI-ESM-LR 2011; URL: ht.... "	Y	A reference of the model.
`product`	M	output	Y	
`references`	M	ECHAM6: Stevens, ...	N	Citations
`leap_year`	M	2000 (Integer)	N	Can be specified if a specific year is skipped independent of the calendar
`leap_month`	M	11 (Integer)	N	Can be specified if a specific month is skipped independent of the calendar
`institude_id`	U	MPI-M	Y	
`institution`	U	"Max Planck Institute for Meteorology"	Y	
`contact`	U	cmip6-mpi-esm@dkrz.de	Y	Mail adress of data creator.

B.4. Attributes recognized or required by the CORDEX standard

Global attribute.	**Associated with topic**	**Example** Default type is String, otherwise mentioned.	**Necessary information**	Explanation
`project_id`	E	CMIP5	Y	Value must be equivalent to the project which can be found in the *MIP-table* passed as argument 1.
`experiment_id`	E	amip	Y	
`realization`	E	1 (Integer)	Y	Part of the ensemble member label.
`initialization_method`	E	1 (Integer)	Y	Part of the ensemble member label.
`physics_version`	E	1 (Integer)	Y	Part of the ensemble member label.
`forcing`	E	oz	N	Description of forcing.
comment	E		N	
`parent_experiment_id`	E	piControl	(Y)	Similar to `experiment_id` but for parent
`parent_experiment_rip`	E	r1i1p1	(Y)	Parent experiment ensemble member label.
`branch_time`	E	50 (Double)	(Y)	Branch time in parent.
`model_id`	M	MPI-ESM	Y	
`source`	M	"MPI-ESM-LR 2011; URL: ht.... "	Y	A reference of the model.
`product`	M	output, model-output	Y	

`references`	M	ECHAM6: Stevens, ...	N	Citations
`leap_year`	M	2000 (Integer)	N	Can be specified if a specific year is skipped independent of the calendar
`leap_month`	M	11 (Integer)	N	Can be specified if a specific month is skipped independent of the calendar
`cordex_domain`	M	EUR11	Y	Value represents the model region and grid spacing of the regional model.
`rcm_version_id`	M	v1	Y	Corresponds to the version of the regional climate model
`driving_model_id`	M	MPI-ESM-LR	Y	Value is the global model name provides the forcing data for the regional model.
`institude_id`	U	MPI-M	Y	
`institution`	U	"Max Planck Institute for Meteorology"	Y	
`contact`	U	cmip6-mpi-esm@dkrz.de	Y	Mail adress of data creator.

B.5. Attributes recognized or required by the CMIP6 standard

Global attributes. Default type is String, otherwise mentioned	**Associated with topic**	**CMIP5 name if different**	**Example**	**Necessary information**	Explanation
mip_era			CMIP6	CMIP6	Phase of CMIP
activity_id	E	project_id	CMIP	Y	One of the registered MIPs
experiment_id	E		piControl	Y	
experiment	E		pre-industrial control	Y	Description of `experiment_id`.
realization_index (integer)	E	realization	1	Y	Part of the ensemble member label.
initialization_index (integer)	E	initialization_method	1	Y	Part of the ensemble member label.
physics_index (integer)	E	physics_version	1	Y	Part of the ensemble member label.
forcing_index (integer)	E	forcing	1	Y	Part of the ensemble member label.
variant_info	E		CO2 forcing from 10 2017.	N	Comment on ensemble member, e.g. info about forcing.
comment	E		N		
sub_experiment_id	E		none	Y	Used for hindcast experiments, usually set to "none".
sub_experiment	E		none	Y	Description of `sub_experiment_id`.
parent_activity_id	E		CMIP	(Y)	Similar to `activity_id`but for parent
parent_experiment_id	E		piControl-spinup	(Y)	Similar to `experiment_id` but for parent

parent_source_id	E		MPIESM-1-2-HR	(Y)	Similar to `source_id` but for parent
parent_time_units	E		"days since 1850-01-01 00:00:00"	(Y)	Similar to `required_time_units` but for parent
parent_variant_label	E	parent_experiment_rip	r1i1p1f1	(Y)	Similar to `variant_label` but for parent
branch_method	E		-	(Y)	Procedure of branching members from spin-ups or parents.
branch_time_in_child	E		40	(Y)	Time value relative to `required_time_units`.
branch_time_in_parent	E	branch_time	10	(Y)	Time value relative to `required_time_units` of parent
grid	G		Regridded on 2 degree	Y	Information about the grid defined as `grid_label`
grid_label	G		gr	Y	Grid identifier. gn is the native grid, gr would be regridded.
nominal_resolution	G		50km	Y	"Rounded" mean over all grid points of the longest distance of grid point vertices distances.
source_id	M	model_id	MPIESM-1-2-HR	Y	One of the registered models. Typical format is: <name>-<version_label>- <resolution>. Does not contain a '_' in order to have a clear file name and file path structure.
source	M		MPIESM-1-2-HR(2017)...	Y	Long description of the model.
source_type	M		AOGCM	Y	Short form of which submodles (atmosphere model, ocean model, ...) took part in the simulations.
references	M		Wachsmann (2017)	N	
institution_id	U	institude_id	MPI-M	Y	One of the registered institutions.
institution	U		Max-Planck-Institute for Meteorology, Hamburg 20146, Germany	Y	Long form of `institution_id`.
license	U		CMIP6 model data produced by DKRZ is licensed ...	Y	Must match a long regular expression.
contact	U			N	

C. Script to install cdo with CMOR support on a unix system

```
#!/bin/sh

#Choose your installation directory HOME:
HOME=/home/

#Download the packages zlib-1.2.8, hdf5-1.8.13, expat-2.2.0, udunits-2.2.20,
# uuid-1.6.2, netcdf-4.4.1.1, jasper-1.900.1, grib_api-1.14.4-Source
# and, of course, cmor2_v292 and cdo-1.8.0rc5 to $HOME

cd zlib-1.2.8/
./configure --prefix=${HOME}
make; make check; make install
cd ../

cd hdf5-1.8.13/
./configure --with-zlib=/${HOME} --prefix=${HOME} CFLAGS=-fPIC
make; make check; make install
cd ../

cd expat-2.2.0/
./configure --prefix=${HOME} CFLAGS=-fPIC
make; make check; make install
cd ..

cd udunits-2.2.20/
CPPFLAGS=-I${HOME}include LDFLAGS=-L${HOME}lib
./configure --prefix=${HOME} CFLAGS=-fPIC
make; make check; make install
cd ..

cd uuid-1.6.2/
./configure --prefix=${HOME} CFLAGS=-fPIC
make; make check; make install
cd ..

cd netcdf-4.4.1.1/
CPPFLAGS=-I${HOME}include LDFLAGS=-L${HOME}lib
./configure --prefix=${HOME}
--enable-netcdf-4 CFLAGS=-fPIC
make; make check; make install
cd ..

cd jasper-1.900.1/
./configure --prefix=${HOME} CFLAGS=-fPIC
make; make check; make install
cd ..
cd grib_api-1.14.4-Source
./configure --prefix=${HOME} CFLAGS=-fPIC --with-netcdf=${HOME} --with-jasper=${HOME}
make; make check; make install
cd ..

cd cmor2_v292/
CFLAGS=-fPIC CPPFLAGS=-I${HOME}include LDFLAGS=-L${HOME}lib
./configure --prefix=${HOME}local
--with-udunits2=${HOME} --with-uuid=${HOME} --with-netcdf=${HOME}
make
make install
cd ..

cd cdo-1.8.0rc5
CPPFLAGS="-I${HOME}include -I${HOME}include/cdTime" LDFLAGS="-L${HOME}lib"
./configure --prefix=${HOME}
--with-cmor=${HOME}local LIBS="-L${HOME}/lib -lnetcdf -ludunits2 -luuid"
--with-netcdf=${HOME} --with-jasper=${HOME} --with-hdf5=${HOME}
--with-grib_api=${HOME} --with-udunits2=${HOME}
make -j8
cd ..
```