

The CDO CMOR operator

Produce CMIP-compliant climate model output
April 2017

Fabian Wachsmann, Stephanie Legutke, Joerg Wegner
Deutsches Klimarechenzentrum (DKRZ)

Contents

1. Introduction	3
2. Installation	4
2.1. CDOs with CMOR2 for CMIP5	4
2.1.1. DKRZ system	4
2.1.2. Local installation	4
2.2. CDOs with CMOR3 for CMIP6	4
3. Operator set-up	5
3.1. Main control	5
3.2. Temporal and spatial description	6
3.3. Global attributes	7
3.4. Variable attributes	8
3.4.1. Variable mapping with table file	10
3.4.2. Example	12
3.5. Special requests	12
3.5.1. Character coordinates	12
3.5.2. Scalar z-coordinate	12
3.5.3. Hybrid sigma-pressure coordinates	13
3.6. Append mode	13
3.6.1. Examples	14
3.7. Internal procedures	14
3.7.1. Temporal boundaries	14
3.7.2. Spatial boundaries	15
3.8. <code>cdo cmor</code> with CMOR 3	15
A. Table of all keywords except global attributes	16
B. Table of all global attribute keywords	17
C. Script to install <code>cdo</code> with CMOR support on a unix system DYNAMIC?	18

1. Introduction

The Climate Data Operator (**CDO**) software is a collection of operators for standard processing of climate and forecast model data. Here, the single operator **cdo cmor** is documented which demands a separate explanation because of its many options to rearrange model output and add Metadata. It represents 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 NetCDF files that fulfill the requirements of many of the climate community's standard model experiments. The output resulting from CMOR is "self-describing" and facilitates analysis of results across models.

CMOR functions feature a specific processing and require much input. The objective of **cdo cmor** is to provide an easy interface to CMOR in order to automate and simplify its usage. **cdo cmor** can guarantee the right CMOR configuration by calling the CMOR functions internally in the right order in which CMOR labels and input attributes are inserted correctly. A reduction of input can be achieved by exploiting the information gained by the climate data interface CDI. The final **cdo cmor** has a minimum requirement of one argument and an input file to apply CMOR accurately

Different versions of the CMOR library have been published over the years. The operator is based on CMOR version 2 which is appropriate to build CMIP5 compliant netcdf model output. The data standard of CMIP5 is fixed so that the project's database could be used to validate and improve the operator's functionality. Besides, there will be no update to another CMOR 2 version which guarantees a stable source package.

This documentation is about the operator which uses CMOR version 2. In next **CDO** releases, the operator will be upgraded by the support for more recent CMOR versions in order to achieve the long term goal of facilitating the preparation of project compliant output for all CMIP phases and other projects.

The **CDO** operator **cmor** was developed at the DKRZ and MPI for Meteorology and will be enhanced at the DKRZ for the CMIP6 project.

2. Installation

Note: Following instructions provide access to a **CDO** test version!

The recent official CDO release does not contain the **cdo cmor** operator because it is still in a test phase within CMIP6. Therefore, the following instructions explain how to access and, if needed, how to install the CDO test version on a DKRZ system and locally on a unix computer. The operator is developed within the CMIP6 project and a release in the official CDOs is aimed. Users of the test version are informed when new updates are available.

2.1. CDOs with CMOR2 for CMIP5

A first tagged CDO test version with support for CMOR version 2.92 is provided at https://svn.dkrz.de/mad/Model/cmor-support/tags/cdo_03-31-2017/ and can be downloaded after a registration on inquiry. The installation of CMOR support for CDO is complex because several packages used by CDO and CMOR need to be combined. It is highly recommended to use the operator on the DKRZ system (mistral) where active user support can be given.

2.1.1. DKRZ system

The provided test version of cdo is installed on mistral in work and the exe file is: [/work/bm0021/cdo_03-31-2017/cdo-git/src/cdo](#). In [/work/bm0021/cdo_03-31-2017/](#), a script for the installation in a user-defined directory is given.

2.1.2. Local installation

An example script how an installation of the downloaded test version can work on a local computer is given in Appendix C. It requires downloading of other packages. Note that this way of installing the version results in full CDO support for GRIB and NETCDF file formats.

2.2. CDOs with CMOR3 for CMIP6

CMOR version 3 can neither be used in the recent CDO release nor in the CDO test version. Therefore:

Note: Conversion for CMIP6 model output not yet enabled!

Once a stable CMOR version 3 is available, the implementation of CMOR version 3 will be developed.

3. Operator set-up

This chapter introduces the operator set-up and all adjustable options. The main control elements are explicated at the beginning to show the usage of the program interface. Since required information is often missing in the input file, in the next sections it is explained how additional information in terms of temporal and spatial decription, global and variable attributes can be easily and effectively delivered to the operator.

Since CMIP requested climate data covers an enormous range of data types across experiments and variables, **cdo cmor** analogously provide many options to enable a correct processing for each of these special requests. The usage of these options are given for each case in the subsequent section.

cdo cmor enables appending of output to an existing chunk. This special append mode is explained in a further section because of its special control mechanism. Finally, the methods of automatic background calculations in internal procedures of **cdo cmor** are introduced.

3.1. Main control

A minimum input when calling **cdo cmor** consists of specifications of a file name (relative or absolute path) of the used *MIP-table* as the first argument and the input file *infile*:

```
cdo cmor,$MIP-table infile          #abstract form
cdo cmor,Dir/CMIP5_Amon test.nc     #example
```

In this case, all necessary information for converting the infile with cmor to a compliant format needs to be available as global and variable attributes in infile. This case is only possible if infile is a NetCDF file. However, there are several ways to pass further information to the operator which are explained beneath.

In contrast to operators which create an outfile, the outfile name must not be specified because it is always generated by CMOR according to the project. The first argument is always the MIP-table.

If only a subset of variables shall be processed, variables can be selected by using a *key=values* pair with the keyword *cmor_name* and values as a comma separated list in a form that:

```
cdo cmor,MIPtable,cmor_name=var1,var2 infile          #abstract form
cdo cmor,Dir/CMIP5_Amon,cmor_name=tas,uas test.nc     #example
cdo cmor,Dir/CMIP5_Amon,cn=tas,uas test.nc            #example 2
```

#example 2 shows a variation with *cn* as a short keyword name for *cmor_name*. For every keyword which is not a global attribute a short name exist. For a succesfull processing of the previous example, the following listed information is required and needs to be given in infile 'test.nc':

- Spatial information, e.g. a grid with coordinates 'longitude' and 'latitude'.
- Temporal information, especially calendar and time units.
- All global attributes:
 - project_id, experiment_id, model_id, source, institute_id, contact, institution
- All attributes of the variables listed in *cmor_name*:
 - units

In general, not all required information is available from infile. Additional global attributes which are not deposited in infile can be collected in files denoted via `info` (a commandline registration for global attributes is locked). Keywords `drs` and `drs_root` determine whether a DRS directory structure is built and in which path, respectively. These general configuration keywords controlling in- and output are comprehensively presented in Table 3.1.

Table 3.1.: Main keywords to control in- and output of `cdo cmor`

Name	Short name	Value description	Default
<code>cmor_name</code>	<code>cn</code>	Strings. Values are cmor names of variables which should be processed. These names must be in the <i>MIP-table</i> given as argument 1. In general, variables in infile whose names agree with the specified cmor_names are picked. <i>Consider that cmor_name is often neither out_name nor standard_name nor long_name.</i>	If no cmor_name is denoted, infile variable names are interpreted as cmor_names and all variables from infile will be processed. If one of these variables has a name which is not denoted as a cmor_name in the <i>MIP-table</i> given as argument 1, CMOR gives an error.
<code>info</code>	<code>i</code>	Strings. Values are file names similar to <i>MIP-table</i> containing global attributes as <i>key=value</i> pairs. These <i>key=value</i> pairs are given in the file as one pair each line.	If no files are denoted, <code>cdo cmor</code> tries to read: <code>'\$HOME/.cdocmorinfo'</code> . If it does not exist, no file is read and the program continues.
<code>drs</code>	<code>d</code>	Character. Value is either 'y' or 'n'. 'y': Directory structure according to DRS is built. 'n': No directory structure according to DRS is built.	0. If the DRS is built, the publication work flow is more efficient.
<code>drs_root</code>	<code>dr</code>	String. Value is a file name where the directory structure according to the DRS is built.	Working directory

`cdo cmor` gains information from the commandline with highest priority, from info files concerning global attributes respectively from a mapping table file concerning variable attributes and from the infile with lowest priority where the priority decides which information is taken if keywords are repeatedly given in more than one of these access points. I.e., an infile global attribute can be overwritten by the corresponding info file keyvalue which can be again overwritten by a commandline keyvalue.

3.2. Temporal and spatial description

Three model and experiment specific keywords exist for adding temporal and spatial description which are required for `cdo cmor`. They are listed in Table 3.2.

Table 3.2.: Keywords to set required temporal and spatial information. Note that **calendar** is only assignable in info files.

Name	Short name	Value description	Default
grid_info	gi	String. Value is a climate data file which contains grid information. A grid is read in from that file and substitutes all grids from variables in infile.	If no 'grid_info' is stated, grids from input file are taken for the variables, respectively. If no grid information at all is available, it depends on the request whether the operator gives an error or not.
required_time_units	rtu	String. Value has the form '\$Frequency since \$Year-\$Month-\$Day \$Hours:\$Minutes:\$Seconds', for example: 'days since 1979-1-1 00:00:00'. The value is required to build a relative time axis where time values of the records in the infile are calculated as the temporal distance to the specified reference time. Generally, the experiment prescribes this value.	If no value is given, the operator tries to work with the time axis in infile. If no time information at all is available, it depends on the request whether the operator gives an error or not.
calendar (only assignable in info files)	1	String. Value is one of 'standard' ('gregorian'), 'proleptic_gregorian', '360_day', 'noleap' and 'all_leap'. Generally, the model prescribes this value.	If no value is given, the operator tries to work with a potential calendar in infile. If no calendar information at all is available, the default is 'standard'.

These keywords **grid_info**, **required_time_units** and **calendar** have in common that they set spatial and temporal axes information respectively for all variables which should be processed at once. Therefore, they can also be set in info files and will be processed similar to global attributes internally. If all variables are computed by the same model and designed for one experiment (which is in general the case), no problems referring to these keyvalues should occur during a conversion of a desired subset of variables because **grid_info** and **calendar** are model specific attributes and **required_time_units** is prescribed by the experiment.

Note that the **grid_info** file cannot be used in a operator chain because, during a chain, two climate data files cannot be open at the same time.

3.3. Global attributes

The necessary global attributes are given in Table 3.3. One important reason why they are mandatory is that they are part of the Data Reference Syntax (DRS) which is used to build the file path for the output.

The attribute **project_id** plays a special role because it affects the importance and restrictions of other global attributes. For particular projects, e.g. 'CMIP5', several global attributes are restricted to a 'Controlled Vocabulary' (CV) which is predefined by the project and used by CMOR to control the attributes. The **project_id** must be equivalent to the project which can be found in the *MIP-table* given as argument 1.

Table 3.3.: Required global attributes to start **cdo_cmor** independently of the project. For particular projects, some of the listed attributes are restricted to a 'Controlled Vocabulary' (CV) which is highlighted in the **Explanation** row with : 'CV'.

Keyword (CMIP6 Name)	Value examples	Explanation
project_id (activity_id)	CMIP5	Value must be equivalent to the project which can be found in the <i>MIP-table</i> given as argument 1.
experiment_id	amip	CV
model_id (source_id)	MPI-ESM	CV
source	"MPI-ESM-LR 2011; URL: ht.... "	A reference of the model.
institute_id (institution_id)	MPI-M	CV
institution	"Max Planck Institute for Meteorology"	CV
contact	"cmip5-mpi-esm@dkrz.de"	-

For different projects, **cdo_cmor** requires additional global attributes to start. They are shown in Table 3.4.

Table 3.4.: Additional required global attributes to start **cdo_cmor** for requests of different projects. Attributes are restricted to a controlled vocabulary when 'CV' is denoted in the **Explanation** row.

Keyword	Required from projects	Value examples	Explanation
member	CMIP5 ⁺ , CORDEX	rlilp1	The format of member is "r%di%dp%d" where %d is an integer respectively. The integers after 'r', 'i' and 'p' represent the realization method, initialization method and physics version respectively.
product	CMIP5 ⁺ , CORDEX	output	CV
cordex_domain	CORDEX	EUR11	CV. Value represents the model region of the regional model.
driving_model_id	CORDEX	MPI-ESM	CV. Value is the model which is used to force the regional model at its boundaries.

We highly recommend to assign these global attributes shown in Table 3.2, Table 3.3 and Table 3.4 to three different Info files categorized by topics Model, Experiment and User (see Appendix). Therefore, the user can be sure to adapt all corresponding attributes when changing one of the topic elements.

3.4. Variable attributes

If infile data contains variables correctly named with **cmor_names** and provided with **units** and **cell_methods** attributes which agree with the CMIP-requested ones, this chapter can be omitted. However, most of the infiles contains rather raw data with variables without attributes and names. In that case, it can be useful to (re-)name the variable read from infile as the fitting **cmor_name** and add necessary variable attributes. This procedure is called 'mapping' from now on.

In general, **cdo cmor** provides two ways to map the input file variable to the one fulfilling all requirements for **cdo cmor**. In the interactive way, only one variable can be mapped in each operator call. The other one uses a table file and is explicated in the next subsection.

Besides **cmor_name**, **cdo cmor** provides two additional variable attributes which act as substitutional variable selectors, **name** and **code**. If one of them is found in the commandline, **cdo cmor** selects the variable from infile via **name** respective **code** and not via **cmor_name** anymore. The value for **code** is the GRIB-code, a three digit integer, of the infile variable one likes to process. The selected variable, however, will be written in the outfile with the name specified via **cmor_name**. To denote both, key name and key

code, is neither reasonable nor allowed. If more than one value is found for one of these three keywords, only the first is processed. If a variable selection key is denoted, a `cmor_name` **must** also be specified.

```

cdo cmor,MIP-table,cmor_name=outname infile          #select by outname
                                                    #output name is outname
cdo cmor,MIP-table,name=inname,cmor_name=outname infile #select by inname
                                                    #output name is outname
cdo cmor,MIP-table,code=incode,cmor_name=outname infile #select by incode
                                                    #output name is outname

```

Variable attributes listed in Table 3.5 can be set in the command line. If a subset of variables should be processed and variable attributes are registered in the command line, the operator gives a warning because all requested variables via 'cmor_name' are registered with the same variable attributes which is probably not desired.

Table 3.5.: Mapping table entry keyword (first line), variable selection keywords (second and third lines) and variable attributes (following lines) which can be added. Note that only one selection key should be denoted, otherwise the operator gives an error.

Name	Short name	Value description	Default
<code>mip_table</code>	<code>mtf</code>	String. Value is identical to <i>MIP_table</i> frequency used in the operator call. It is needed in a mapping table to distinguish between two lines containing equivalent <code>cmor_names</code> .	If no keyvalue is given, the operator uses the first line it finds in the mapping table with agreeing <code>cmor_names</code> .
<code>name</code>	<code>n</code>	String. Value is the variable name in infile of the variable requested via ' <code>cmor_name</code> '. If ' <code>name</code> ' or ' <code>code</code> ' is denoted in the command line, only one variable is selected from infile by ' <code>name</code> ' respective ' <code>code</code> '.	If no value is given, no variable renaming is active.
<code>code</code>	<code>c</code>	Three digits Integer. Value is the variable code of the variable requested via ' <code>cmor_name</code> ' in infile. If ' <code>name</code> ' or ' <code>code</code> ' is denoted in the command line, only one variable is selected from infile by ' <code>name</code> ' respective ' <code>code</code> '.	If no value is given, no variable renaming is active.
<code>units</code>	<code>u</code>	String. Value must be readable by library ' <code>udunits</code> '(LINK), e.g.: ' <code>W m-2</code> '. CMOR uses this library to convert units if they do not agree with the request.	If no value is registered, units are taken from infile. If no units information is available, the operator gives an error.
<code>cell_methods</code>	<code>cm</code>	Character or first character of string. Value is one of ' <code>m</code> ', ' <code>p</code> ', ' <code>c</code> ' and ' <code>n</code> ' which stands for ' <code>mean</code> ', ' <code>point</code> ', ' <code>climate</code> ' and ' <code>none</code> '. This is the aggregation method used for the chosen variable in infile. Its value is necessary to register the correct time axis. ' <code>Mean</code> ' is used for temporal averages, ' <code>point</code> ' for observations, ' <code>climate</code> ' for and ' <code>none</code> ' for static fields.	The default is mean i.e. ' <code>m</code> '.
<code>positive</code>	<code>p</code>	Character. Value is one of ' <code>u</code> ', ' <code>d</code> ' and ' <code>'</code> ' (blank). This attribute enables CMOR to switch the sign of a directed variable, e.g. radiation flux, if the request demands it. ' <code>u</code> ' stands for ' <code>upward</code> ', ' <code>d</code> ' for ' <code>downward</code> ' and a blank symbolizes an undirected variable.	The demand is blank. However, if a directed variable is registered, one of ' <code>u</code> ' and ' <code>d</code> ' is required - otherwise the operator gives an error.
<code>variable_comment</code>	<code>vc</code>	String. Value gives additional information about the variable.	If no keyvalue is given, no comment will be in output and the program continues.

Important note concerning `cell_methods`: The requested `cell_methods` must be equivalent to the aggregation method which was used to create the infile. `cell_methods` cannot be changed, for example, if the output is averaged over time but `cell_methods: point` is requested.

3.4.1. Variable mapping with table file

Cdo cmor offers a way to map several variables 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 of the format of a fortran namelist. Each line of this file is related to one variable and consists of attributes one likes to provide the variable with. A line must begin with the keyword `¶meter` by definition. An example of three lines of a mapping table is given in the following.

```
&parameter out_name=tas      name=temp2    code=167  p=" "  units="K"  cell_methods="m" /
&parameter out_name=sos      name=sss     code=016  units="psu"  cell_methods=m /
&parameter out_name=msftbarot name=psitro   code=027  units="kg s-1" cell_methods=m /
```

Consider for preparation of a mapping table:

1. Each line must begin with the entry keyword `¶meter`.
2. Both, the order of further keywords in each line and blanks between value and next keyword are irrelevant.
3. If a value contains blanks, it must be surrounded by `''`.
4. In contrast to the interactive mapping, one can also denote both, name and code in the table.

Usually, the attribute entries for the mapping table are keywords `cmor_name` and `mip_table` which, besides, together identify a unique `cmor` variable. The mapping table line which contains the `cmor_name` specified via keyvalue is searched. If no line with the corresponding `cmor_name` is found, the mapping table is not processed and the operator continues. Otherwise, if a `mip_table` keyvalue is in the found line it has to agree with *MIP-table* given as argument 1. That means, there can be multiple lines for the same `cmor_name` because it may require different `units` or `cell_methods` attributes in another `mip_table`. If the `mip_table` value disagrees, the search continues. If no `mip_table` is denoted in the first line of agreeing `cmor_names`, the line is taken for mapping.

If a suitable mapping table line with agreeing `cmor_name` and, if applicable, `mip_table` is found, the corresponding infile variable needs to be detected. Therefore, variable selectors `name` and `code` are searched in the found line and compared with infile variables. Keyword `name` has higher priority: If a variable in infile has the corresponding name all further attributes of this line are assigned to this variable. However, if no `name` is found in a line of the mapping table, the other selector attribute `code` is compared with variable codes in infile. If no selector attribute is denoted or no selector attribute corresponds to an infile variable, the line in the mapping table will be ignored. Two special cases are discussed in the following.

Case: Commandline mapping + Mapping table

Note that a combination of a commandline mapping and a mapping table file is possible: If, in addition to a mapping table, a variable selection key (`name` or `code`) plus a `cmor_name` is registered in the command line, only this variable is mapped and all attributes from the mapping table are also applied on the variable. The line in the mapping table is again selected by `cmor_name`. However, value of keyword `name` found in the mapping table line is ignored and the variable from infile is taken by the `name` specified in the command line. Therefore, the combination of both mapping methods can be reasonable e.g. if the origin model output variable name changed after the mapping table was constructed.

Case: Mapping table but no cmor_name

If all variables in infile should be processed and no `cmor_name` is specified, the mapping table entry keyword is no longer `cmor_name` but `name`, with higher priority, and `code`, with lower priority. That is, if a line contains a variable selector keyword which agrees with the infile variable attribute, the line is used for mapping.

All external files, the `mapping_table`, the `grid_info` file as well as the *MIP-table* file can be configured either via the filename inclusive directory path on the whole specified with one keyword or in combination with corresponding keywords `mapping_table_dir`, `grid_info_dir` and `mip_table_dir` respectively. The operator builds the complete file name with these keywords - in case of the *MIP-table* in addition with `project_id` so that the call `'cdo cmor,Amon,...'` can be sufficient.

3.4.2. Example

3.5. Special requests

Cdo cmor enable the conversion of variables with special features concerning scalar, character or vertical hybrid coordinates. They are explicated in detail in the following sections. An overview about the additional keywords one can configure to handle these special variable conversions is given in Table 3.6. The keywords are processed similar to variable attributes and can be specified in both, the command line and the mapping table. Again, consider that configurations specified in the command line are valid for all in the operator call requested variables.

These special requests are often associated with *cmor labels*. This expression means that CMOR requires exactly the special name denoted in the MIP-table instead of a standard name or out_name to run properly. There is no other way to look up the label to process this request properly.

Table 3.6.: Special keywords in **cdo_cmor** for special request treatment

Name	Short name	Value description	Default
scalar_z_coordinate	szc	String. Value has the shape: '(\$axisname)-(\$axisvalue)', where \$axisname is the requested axis label for a scalar z-coordinate and \$value the deviant value, e.g.: 'height2m.1.5'.	If a scalar z-coordinate is requested but not registered, the requested value of this coordinate is automatically taken.
character_axis	ca	String. Value is a character axis, whose values needs to be registered as well as a global attribute (must not stand in mapping table). <i>At this development stage of the operator, cdo cmor only processes character axes 'basin', 'oline' and 'vegtype' because character axes cannot be transmitted to CMOR at any stage of the process which impedes a general handling.</i>	No default

3.5.1. Character coordinates

Special variables feature a requested axis whose values are not numerical. E.g., ocean model data can be desired for ocean means so that one axis would correspond to something like 'oceans'. The interested reader is pointed to the CF-conventions which define two standard names for that axis type: 'region' and 'area_type'.

Such an axis can be registered for **cdo cmor** with the keyword: **character_axis** and a connected second keyword to define the values of the axis. The value of **character_axis** must be the cmor label of an axis of the requested variable. The second keyword equals this axis label and contains all the axis coordinates: If **character_axis=basin** is denoted, a keyvalues pair **basin=atlantic,pacific,...** must be denoted in an info file similar to a global attribute.

In case the arrays for each character coordinate are saved as individual variables, one can denote more variable names respective codes in the mapping table line of the resulting **cmor_name**. This indicates that the operator has to merge these variables as a character coordinate. If dimensions of all variables agree, the dimension which contains only one value is converted to the character axis. *This is a work around as long as character coordinates are not build in the CDI data model. However, a solution is in progress.*

3.5.2. Scalar z-coordinate

Some variables requested for one coordinate on the z-axis have a special output design. Instead of exhibiting a dimension for this one-value-axis, the output file of the processed variable contains a 'description' variable

pointing at the corresponding coordinate value. A header comparison for illustration is given as an example: (missing).

The requested coordinate value is identifiable from the `cmor` label of this axis which is given in the MIP-table. E.g., 'tas' has a dimension 'height2m' which corresponds to a scalar z-coordinate height where '2m' is the value. CMOR automatically writes the description variable (height) in the output with the corresponding requested value (2m).

However, for some models this value might need to be changed because, e.g., it calculates near-surface variables in a different height. In this case, the `szc` must be configured. The value of `szc` must have the form '(\$axisname)-(\$axisvalue)', e.g. `szc=height2m_1.5`. In many cases, the value must be within a range prescribed by the project with a minimum and a maximum.

3.5.3. Hybrid sigma-pressure coordinates

The registration of a hybrid vertical coordinate like sigma-pressure requires the delivery of some `zaxis` parameter as well as the additional variable surface pressure which will be also written in output. These parameters and the variable needs to be saved in the infile in a form that CDI can process them. The surface pressure variable is identified via variable name 'ps' from infile. If a `mapping_table` is delivered, the `name` which is in the line with `cmor_name=ps` is used to find the correct infile variable. If `ps` is not available the operator gives an error.

3.6. Append mode

If a climate model produces output for each simulation year of a longer period of years, one may want to use **cdm cmor** piece by piece every time new output is available. Therefore, one can use the append mode of CMOR to append a file to an already CMOR-converted file called chunk. **cdm cmor** facilitates to set keywords `output_mode`, `last_chunk` and the maximal chunk size `max_size`. In standard replace output mode, a new file is always generated which replaces an old file when indicated. If `output_mode` is set to 'a' for append mode, **cdm cmor** tries to open a chunk which should be configured via `last_chunk`.

If the append mode is consecutively used several times in operational usage, the file name continuously changes which makes the specification of `last_chunk` difficult. Therefore, the name of the outfile is written into a file of the form: 'APPEND.FILE_ cmor_name)_ mip-tab-freq_ model_id_ experiment_id_ member).txt' in the working directory if **cdm cmor** is used in append mode. E.g.: 'APPEND.FILE_ tas_ mon_ MPI-ESM_ Amon_ rli1p1.txt'. When **cdm cmor** is started in append mode and with `drs=y` again, the file name saved in 'APPEND.FILE_ tas_ mon_ MPI-ESM_ Amon_ rli1p1.txt' is used as a default chunk. If no chunk is valid, the output mode switches to replace and the operator continues.

In the process of appending, it is ensured that the temporal gap between the last time value of the chunk and the new file is in order of the frequency. Another constriction is that the finally appended outfile should not contain more than an adjustable size. The corresponding keyword is `max_size` and is 4Gb per default. This is due to the fact that the publication server can have requests on the file size.

Table 3.7.: Keywords to control the output mode of **cdo_cmor**

Name	Short name	Value 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. The adjustment of the append mode is explained in the text.	'r' for replace
last_chunk	lc	Strings. Values are chunk file names corresponding to the order of requested variables via 'cmor_name'.	For CMIP5 and 'drs'=1, file names are built with attributes in the form: 'APPEND_FILE_ (\$cmor_name)_ (\$mipfreq)_ (\$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 valid in append mode.	4

3.6.1. Examples

3.7. Internal procedures

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

3.7.1. Temporal boundaries

If `cell_methods` is 'mean', CMOR demands for time bounds. If, in that case, no time bounds are configured, **cdo cmor** calculates them based on the time values in infile 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.

For requests with subdaily frequencies and `cell_methods=mean`, it is important that the time value from infile is definitely correct because there is no explicit rule to calculate time bounds for 3hr frequency. I.e., CMOR will not move the time value.

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 the 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.

3.7.2. 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, respectively, 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 longitudes 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, longitudes 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 (latitudes) can be derived by averaging adjacent half-longitudes (-latitudes). I.e., the grid cell corner longitudes (latitudes) are the half-longitudes-on-half-latitudes (half-latitudes-on-half-longitudes). If the absolute difference between two adjacent longitude values is bigger than 180 degrees, 180 degrees are added (subtracted) 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.

3.8. cdo cmor with CMOR 3

- not possible yet

A. Table of all keywords except global attributes

Function	Key name Italic: Internally processed like a global Attribute	Key short name	Type	Necessary either via key or Ifile information (Y = yes, N = no) and restrictions	Default if no key is configured (- = irrelevant)
Variable selector	cmor_name	cn	Strings	N	-
Variable selector	name	n	String	N	-
Variable selector	code	c	Three digits integer	N	-
Main control	info	i	Strings	N	/\$HOME /.cdcmorinfo
Main control	drs	d	Character	N	yes
Main control	drs_root	dr	String	N	Working directory
Temporal/spatial description	<i>grid_info</i>	gi	String	Y and must fit to variable array	Ifile variable grid
Temporal/spatial description	<i>required_time_units</i>	rtu	String	Y and format controlled	Ifile time units
Temporal/spatial description	<i>calendar</i>	l	String	Y and 5 options	Ifile calendar
Operational control	<i>mapping_table</i>	-	String	N	N
Operational control	<i>mapping_table_dir</i>	-	String	N	N
Operational control	<i>mip_table_dir</i>	-	String	N	N
Operational control	<i>grid_info_dir</i>	-	String	N	N
Operational control	output_mode	om	Character	N and 2 options	Replace mode
Operational control	last_chunk	lc	Strings	N	For CMIP5
Operational control	max_size	ms	Integer	N and in Gb	4Gb
Variable attribute	units	u	String	Y and UD_unit compatible	Ifile variable units
Variable attribute	cell_methods	cm	String	N and 4 options	mean
Variable attribute	positive	p	Character	N and 3 options	blank
Variable attribute	variable_comment	vc	String	N	N
Special variable attribute	scalar _z.coordinate)	szc	String and form controlled	N	N
Special variable attribute	character_axis	ca	String and 3 options	N	N

B. Table of all global attribute keywords

Global attributes. Default type is String, otherwise mentioned. <i>Italic</i> : Not a glob. att. but denotable on the same level.	Associated with: E: Experiment, M: Model, U: User.	CMIP6 changes if available	Necessary either via key or Ifile information (Y = yes, N = no, D = Depends on project) and restrictions (CV: Controlled vocabulary)	Default if no information via key or Ifile is available
<i>required_time_units</i>	E		Y	N
project_id	E	activity_id	Y	N
experiment_id	E		Y and CV	N
member	E	Format with forcing	D and format controlled	N
product	E		D and CV	N
cortex_domain	E		D and CV	N
driving_model_id	E		D and CV	N
driving_experiment_name	E		D and CV	N
history	E		N	N
parent_experiment_id	E		N and CV	N
parent_experiment_rip	E		N and CV	N
forcing	E		N	N
branch_times (Format: Double)	E	branch_time_in_parent	N	0.0
model_id	M	source_id	Y	N
source	M		Y	N
<i>mapping_table</i>	M		N	N
<i>mapping_table_dir</i>	M		N	N
<i>grid_info</i>	M		N	N
<i>grid_info_dir</i>	M		N	N
<i>calendar</i>	M		Y and 5 options	"standard" (gregorian)
rcm_version_id	M		N	N
references	M		N	"No references available for \$model_id"
institute_id	U	institution_id	Y and CV	N
institution	U		Y and CV	N
contact	U		Y and CV	N
<i>mip_table_dir</i>	U		N	N

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

```
#!/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 ..
```