
ESMValTool User's and Developer's Guide

Release 2.15.0.dev34+gc64309e13

ESMValTool Development Team

May 08, 2026

ESMVALCORE

I	Basic documentation schema	3
II	Learning resources:	7
III	How to contribute	11
IV	Get in touch!	15
1	Getting started	19
2	Example notebooks	57
3	The recipe format	75
4	Diagnostic script interfaces	129
5	Development	133
6	Contributions are very welcome	147
7	How-to guides	157
8	Reference guides	161
9	ESMValCore API Reference	167
10	Changelog	305
V	Indices and tables	355
	Python Module Index	359
	Index	361

ESMValTool is a community diagnostics and performance metrics tool for the evaluation of Earth System Models (ESMs) that allows for routine comparison of models and observations. It includes a large collection of community recipes and observation data formatters to CMOR standards.

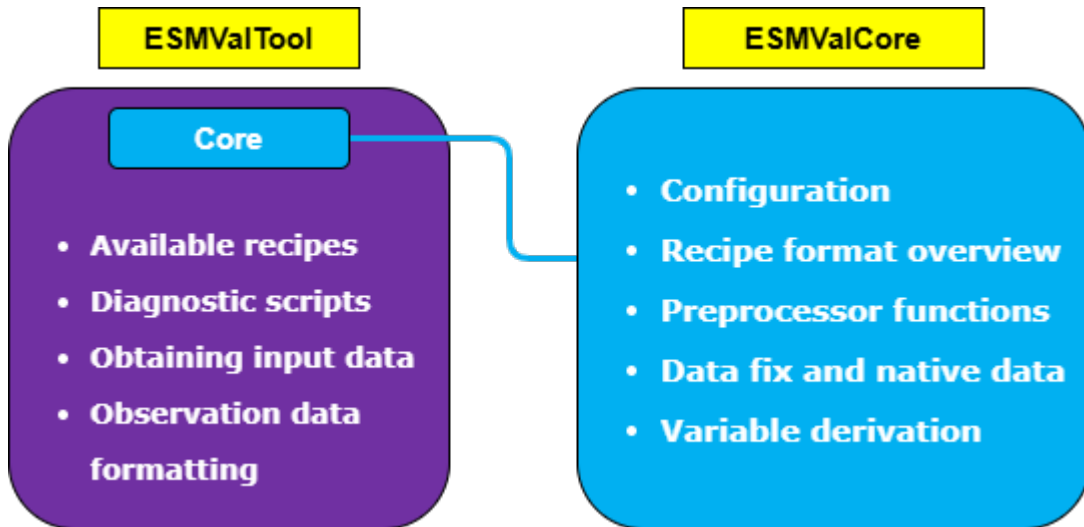
ESMValCore is a software package which provides the core functionality for ESMValTool. It is a workflow to find CMIP data, and apply commonly used pre-processing functions.

To get a first impression of what ESMValTool and ESMValCore can do for you, have a look at our blog posts [Analysis-ready climate data with ESMValCore and ESMValTool: Recipes for solid climate science](#).

Part I

Basic documentation schema

This gives a brief idea of topics in each of the packages to help find information. As ESMValTool encompasses ESMValCore there will be some overlap between them. ESMValCore can be used without ESMValTool such as in a Jupyter notebook. For more detailed information, see documentation navigation to the left. Please also see [ESMValTool documentation](#).



Part II

Learning resources:

A carpentries tutorial is available on <https://tutorial.esmvaltool.org>.

A series of video lectures has been created by [ACCESS-NRI](#). While these are tailored for ACCESS users, they are still very informative.

Part III

How to contribute

Refer to ESMValTool contributing to the community for a guide on contributing recipes and diagnostics.

Refer to ESMValCore *Contributing* for information on contributing code.

Part IV

Get in touch!

Contact information is available [here](#).

GETTING STARTED

1.1 Installation

1.1.1 Mamba/Conda installation

In order to install ESMValCore and its dependencies from `conda-forge`, you will first need to install the `mamba` package manager. We recommend using `mamba` instead of `conda` because it is faster and offers the same commands. For a minimal `mamba` installation (recommended) go to <https://mamba.readthedocs.io/en/latest/installation/micromamba-installation.html>.

Tip

It is recommended that you always use the latest version of `mamba`, as problems have been reported when trying to use older versions.

Once you have installed `mamba`, you can install ESMValCore by running:

```
mamba install -c conda-forge esmvalcore
```

It is also possible to create a new `Conda environment` and install ESMValCore into it with a single command:

```
mamba create --name esmvalcore -c conda-forge esmvalcore
```

Don't forget to activate the newly created environment after the installation:

```
mamba activate esmvalcore
```

Of course it is also possible to choose a different name than `esmvalcore` for the environment.

Note

Creating a new `Conda environment` is often much faster and more reliable than trying to update an existing `Conda environment`.

1.1.2 Pip installation

It is also possible to install ESMValCore from `PyPI`. However, this requires first installing dependencies that are not available on `PyPI` in some other way. The list of required dependencies can be found in `pyproject.toml`.

⚠ Warning

It is recommended to use the installation with mamba instead, as it may not be easy to install the correct versions of all dependencies.

After installing the dependencies that are not available from [PyPI](#), install ESMValCore and any remaining Python dependencies with the command:

```
pip install esmvalcore
```

1.1.3 Docker installation

ESMValCore is also provided through [DockerHub](#) in the form of docker containers. See <https://docs.docker.com> for more information about docker containers and how to run them.

You can get the latest release with

```
docker pull esmvalgroup/esmvalcore:stable
```

If you want to use the current main branch, use

```
docker pull esmvalgroup/esmvalcore:latest
```

To run a container using those images, use:

```
docker run esmvalgroup/esmvalcore:stable --help
```

Note that the container does not see the data or environmental variables available in the host by default. You can make data available with `-v /path:/path/in/container` and environmental variables with `-e VARNAME`.

For example, the following command would run a recipe

```
docker run -e HOME -v "$HOME":"$HOME" -v /data:/data esmvalgroup/esmvalcore:stable ~/
↳ recipes/recipe_example.yml
```

with the environmental variable `$HOME` available inside the container and the data in the directories `$HOME` and `/data`, so these can be used to find the configuration, recipe, and data.

It might be useful to define a [bash alias](#) or script to abbreviate the above command, for example

```
alias esmvaltool="docker run -e HOME -v $HOME:$HOME -v /data:/data esmvalgroup/
↳ esmvalcore:stable"
```

would allow using the `esmvaltool` command without even noticing that the tool is running inside a Docker container.

1.1.4 Singularity installation

Docker is usually forbidden in clusters due to security reasons. However, there is a more secure alternative to run containers that is usually available on them: [Singularity](#).

Singularity can use docker containers directly from DockerHub with the following command

```
singularity run docker://esmvalgroup/esmvalcore:stable ~/recipes/recipe_example.yml
```

Note that the container does not see the data available in the host by default. You can make host data available with `-B /path:/path/in/container`.

It might be useful to define a `bash` alias or script to abbreviate the above command, for example

```
alias esmvaltool="singularity run -B $HOME:$HOME -B /data:/data docker://esmvalgroup/
↳esmvalcore:stable"
```

would allow using the `esmvaltool` command without even noticing that the tool is running inside a Singularity container.

Some clusters may not allow to connect to external services, in those cases you can first create a singularity image locally:

```
singularity build esmvalcore.sif docker://esmvalgroup/esmvalcore:stable
```

and then upload the image file `esmvalcore.sif` to the cluster. To run the container using the image file `esmvalcore.sif` use:

```
singularity run esmvalcore.sif ~/recipes/recipe_example.yml
```

1.1.5 Installation from source

Note

If you would like to install the development version of ESMValCore alongside ESMValTool, please have a look at [these instructions](#).

To install from source for development, follow these instructions.

- Install `pixi`
- Clone the ESMValCore Git repository: `git clone https://github.com/ESMValGroup/ESMValCore.git`
- Go to the source code directory: `cd ESMValCore`
- Create the development environment and start a shell in this environment by running: `pixi shell`
- Test that your installation was successful by running `esmvaltool -h`.
- Install the `Pre-commit` hooks by running: `pre-commit install`.

Tip

If you find that solving the environments (i.e. finding out which combination of package versions is compatible and can be installed) is slow, you can add the `--frozen` flag to the commands above to skip the solve step. Add `export PIXI_FROZEN=true` to your `~/.bashrc` file to make this the default behavior.

Tip

To exit the `pixi` environment, run `exit` or press `Ctrl+D`.

1.1.6 Pre-installed versions on HPC clusters / other servers

If you would like to use pre-installed versions on HPC clusters (currently CEDA-JASMIN and DKRZ-Levante), and other servers (currently Met Office Linux estate), please have a look at [these instructions](#).

1.2 Configuration

1.2.1 Overview

Similar to `Dask`, ESMValCore provides one single configuration object that consists of a single nested dictionary for its configuration.

Note

In v2.12.0, a redesign process of ESMValTool/Core's configuration started. Its main aim is to simplify the configuration by moving from many different configuration files for individual components to one configuration object that consists of a single nested dictionary (similar to [Dask's configuration](#)). This change will not be implemented in one large pull request but rather in a step-by-step procedure. Thus, the configuration might appear inconsistent until this redesign is finished. A detailed plan for this new configuration is outlined in [Issue #2371](#).

1.2.2 Specify configuration for `esmvaltool` command line tool

When running recipes via the *command line*, configuration options can be specified via YAML files and command line arguments. The options from all YAML files and command line arguments are merged together using `dask.config.collect()` to create a single configuration object, which properly considers nested objects (see `dask.config.update()` for details). Configuration options given via the command line will always be preferred over options given via YAML files.

YAML files

Configuration options can be specified via YAML files (i.e., *.yaml and *.yml).

A file could look like this (for example, located at `~/ .config/esmvaltool/config.yml`):

```
output_dir: ~/esmvaltool_output
max_parallel_tasks: 1
```

ESMValCore searches for **all** YAML files in **each** of the following locations and merges them together:

1. The directory specified via the `--config_dir` command line argument.
2. The user configuration directory: by default `~/ .config/esmvaltool`, but this location can be changed with the `ESMVALTOOL_CONFIG_DIR` environment variable.

Preference follows the order in the list above (i.e., the directory specified via command line argument is preferred over the user configuration directory). Within a directory, files are sorted lexicographically, and later files (e.g., `z.yml`) will take precedence over earlier files (e.g., `a.yml`).

Warning

ESMValCore will read **all** YAML files in these configuration directories. Thus, other YAML files in this directory which are not valid configuration files (like the old `config-developer.yml` files) will lead to errors. Make sure to move these files to a different directory.

The minimal required configuration for the tool is that you configure where it can find *input data*. In addition to that, you may copy the default configuration file with *top level options*

To get a copy of the default configuration file, you can run the command:

```
esmvaltool config copy defaults/config-user.yml
```

This will copy the file to your configuration directory and you can tailor it for your system, e.g. set the `output_dir` to a path where ESMValTool can store its output files.

Command line arguments

All *configuration options* can also be given as command line arguments to the `esmvaltool` executable.

Example:

```
esmvaltool run --max_parallel_tasks=2 /path/to/recipe.yml
```

Options given via command line arguments will always take precedence over options specified via YAML files.

1.2.3 Specify/access configuration for Python API

When running recipes with the *experimental Python API*, configuration options can be specified and accessed via the `CFG` object. For example:

```
>>> from esmvalcore.config import CFG
>>> CFG['output_dir'] = '~/esmvaltool_output'
>>> CFG['output_dir']
PosixPath('/home/user/esmvaltool_output')
```

Or, alternatively, via a context manager:

```
>>> with CFG.context(log_level="debug"):
...     print(CFG["log_level"])
debug
>>> print(CFG["log_level"])
info
```

This will also consider YAML configuration files in the user configuration directory (by default `~/.config/esmvaltool`, but this can be changed with the `ESMVALTOOL_CONFIG_DIR` environment variable).

More information about this can be found [here](#).

1.2.4 Top level configuration options

Note: the following entries use Python syntax. For example, Python's `None` is YAML's `null`, Python's `True` is YAML's `true`, and Python's `False` is YAML's `false`.

Option	Description	Type	Default value
auxiliary_data	Directory where auxiliary data is stored. ¹	str	~/auxiliary_data
check_level	Sensitivity of the CMOR check (debug, strict, default, relaxed, ignore), see <i>Customizing checker strictness</i> .	str	default
compress_netCDF	Use netCDF compression.	bool	False
configDeveloper	Path to custom <i>Developer configuration file</i> .	str	None (default file)
dask	<i>Dask configuration</i> .	dict	See <i>Default options</i>
diagnostics	Only run the selected diagnostics from the recipe, see <i>Running</i> .	list or str	None (all diagnostics)
download_dir	[deprecated] Directory where downloaded data will be stored. ²	str	~/climate_data
drs	[deprecated] Directory structure for input data. ^{Page 25, 2}	dict	{CMIP3: ESGF, CMIP5: ESGF, CMIP6: ESGF, CORDEX: ESGF, obs4MIPs: ESGF}
exit_on_warning	Exit on warning (only used in NCL diagnostic scripts).	bool	False
log_level	Log level of the console (debug, info, warning, error).	str	info
logging	<i>Logging configuration</i> .	dict	See <i>Logging configuration</i>
max_datasets	Maximum number of datasets to use, see <i>Running</i> .	int	None (all datasets from recipe)
max_parallel	Maximum number of parallel processes, see <i>Task priority</i> . ⁴	int	None (number of available CPUs)
max_years	Maximum number of years to use, see <i>Running</i> .	int	None (all years from recipe)
output_dir	Directory where all output will be written, see <i>Output</i> .	str	~/esmvaltool_output
output_file_type	Plot file type.	str	png
profile_diagnostic	Use a profiling tool for the diagnostic run. ³	bool	False
projects	<i>Project-specific configuration</i> .	dict	See table in <i>Project-specific configuration</i>
remove_preproc	Remove the preproc directory if the run was successful, see <i>Preprocessed datasets</i> .	bool	True
resume_from	Resume previous run(s) by using preprocessor output files from these output directories, see <i>Running</i> .	list of str	[]
rootpath	[deprecated] Rootpaths to the data from different projects. ^{Page 25, 2}	dict	{default: ~/climate_data}
run_diagnostics	Run diagnostic scripts, see <i>Running</i> .	bool	True
save_intermediate	Save intermediary cubes from the preprocessor, see also <i>Preprocessed datasets</i> .	bool	False
search_data	Perform a quick or complete search for input data. When set to quick, search will stop as soon as a result is found. <i>Data sources</i> with a lower value for priority will be searched first. (quick, complete)	str	quick
search_esgf	[deprecated] Automatic data download from ESGF (never, when_missing, always). ^{Page 25, 2}	str	never
skip_nonexistent	Skip non-existent datasets, see <i>Running</i> .	bool	False

1.2.5 Dask configuration

Configure Dask in the dask section.

The *preprocessor functions* and many of the Python diagnostics in ESMValTool make use of the Iris library to work with the data. In Iris, data can be either *real* or *lazy*. Lazy data is represented by *dask arrays*. Dask arrays consist of many small *numpy arrays* (called chunks) and if possible, computations are run on those small arrays in parallel. In order to figure out what needs to be computed when, Dask makes use of a ‘scheduler’. The default (thread-based) scheduler in Dask is rather basic, so it can only run on a single computer and it may not always find the optimal task scheduling solution, resulting in excessive memory use when using e.g. the *esmvalcore.preprocessor.multi_model_statistics()* preprocessor function. Therefore it is recommended that you take a moment to configure the *Dask distributed* scheduler. A Dask scheduler and the ‘workers’ running the actual computations, are collectively called a ‘Dask cluster’.

Dask profiles

Because some recipes require more computational resources than others, ESMValCore provides the option to define “Dask profiles”. These profiles can be used to update the *Dask user configuration* per recipe run. The Dask profile can be selected in a YAML configuration file via

```
dask:
  use: <NAME_OF_PROFILE>
```

or alternatively in the command line via

```
esmvaltool run --dask='{"use": "<NAME_OF_PROFILE>"}' recipe_example.yml
```

Available predefined Dask profiles:

- *local_threaded* (selected by default): use *threaded scheduler* without any further options.
- *local_distributed*: use *local distributed scheduler* without any further options.
- *debug*: use *synchronous Dask scheduler* for debugging purposes. Best used with *max_parallel_tasks*: 1.

To copy these predefined profiles to your configuration directory for further customization, run the command:

¹ The *auxiliary_data_dir* setting is the path to place any required additional auxiliary data files. This is necessary because certain Python toolkits, such as cartopy, will attempt to download data files at run time, typically geographic data files such as coastlines or land surface maps. This can fail if the machine does not have access to the wider internet. This location allows the user to specify where to find such files if they can not be downloaded at runtime. The example configuration file already contains two valid locations for *auxiliary_data_dir* directories on CEDA-JASMIN and DKRZ, and a number of such maps and shapefiles (used by current diagnostics) are already there. You will need esmeval group workspace membership to access the JASMIN one (see *instructions* how to gain access to the group workspace).

Warning

This setting is not for model or observational datasets, rather it is for extra data files such as shapefiles or other data sources needed by the diagnostics.

² This option is scheduled for removal in v2.14.0. Please use *data sources* to configure data finding instead.

⁴ When using *max_parallel_tasks* with a value larger than 1 with the Dask threaded scheduler, every task will start *num_workers* threads. To avoid running out of memory or slowing down computations due to competition for resources, it is recommended to set *num_workers* such that *max_parallel_tasks* * *num_workers* approximately equals the number of CPU cores. The number of available CPU cores can be found by running `python -c 'import os; print(len(os.sched_getaffinity(0)))'`. See *Custom Dask threaded scheduler configuration* for information on how to configure *num_workers*.

³ The *profile_diagnostic* setting triggers profiling of Python diagnostics, this will tell you which functions in the diagnostic took most time to run. For this purpose we use *vprof*. For each diagnostic script in the recipe, the profiler writes a *.json* file that can be used to plot a *flame graph* of the profiling information by running

```
vprof --input-file esmvaltool_output/recipe_output/run/diagnostic/script/profile.json
```

Note that it is also possible to use *vprof* to understand other resources used while running the diagnostic, including execution time of different code blocks and memory usage.

```
esmvaltool config copy defaults/dask.yml
```

Dask distributed scheduler configuration

Here, some examples are provided on how to use a custom Dask distributed scheduler. Extensive documentation on setting up Dask Clusters is available [here](#).

Note

If not all preprocessor functions support lazy data, computational performance may be best with the *threaded scheduler*. See [Issue #674](#) for progress on making all preprocessor functions lazy.

Personal computer

Create a `distributed.LocalCluster` on the computer running ESMValCore using all available resources:

```
dask:
  use: local_cluster # use "local_cluster" defined below
  profiles:
    local_cluster:
      cluster:
        type: distributed.LocalCluster
```

This should work well for most personal computers.

Note

If running this configuration on a shared node of an HPC cluster, Dask will try and use as many resources it can find available, and this may lead to overcrowding the node by a single user (you)!

Shared computer

Create a `distributed.LocalCluster` on the computer running ESMValCore, with 2 workers with 2 threads/4 GiB of memory each (8 GiB in total):

```
dask:
  use: local_cluster # use "local_cluster" defined below
  profiles:
    local_cluster:
      cluster:
        type: distributed.LocalCluster
        n_workers: 2
        threads_per_worker: 2
        memory_limit: 4GiB
```

this should work well for shared computers.

Computer cluster

Create a Dask distributed cluster on the Levante supercomputer using the `Dask-Jobqueue` package:

```
dask:
  use: slurm_cluster # use "slurm_cluster" defined below
```

(continues on next page)

(continued from previous page)

```

profiles:
  slurm_cluster:
    cluster:
      type: dask_jobqueue.SLURMCluster
      queue: shared
      account: <YOUR_SLURM_ACCOUNT>
      cores: 8
      memory: 7680MiB
      processes: 2
      interface: ib0
      local_directory: "/scratch/b/<YOUR_DKRZ_ACCOUNT>/dask-tmp"
      n_workers: 24

```

This will start 24 workers with `cores / processes = 4` threads each, resulting in `n_workers / processes = 12` Slurm jobs, where each Slurm job will request 8 CPU cores and 7680 MiB of memory and start `processes = 2` workers. This example will use the fast infiniband network connection (called `ib0` on Levante) for communication between workers running on different nodes. It is important to set the right location for temporary storage, in this case the `/scratch` space is used. It is also possible to use environmental variables to configure the temporary storage location, if your cluster provides these.

A configuration like this should work well for larger computations where it is advantageous to use multiple nodes in a compute cluster. See [Deploying Dask Clusters on High Performance Computers](#) for more information.

Externally managed Dask cluster

To use an externally managed cluster, specify an `scheduler_address` for the selected profile. Such a cluster can e.g. be started using the [Dask Jupyterlab extension](#):

```

dask:
  use: external # Use the `external` profile defined below
  profiles:
    external:
      scheduler_address: "tcp://127.0.0.1:43605"

```

See [here](#) for an example of how to configure this on a remote system.

For debugging purposes, it can be useful to start the cluster outside of ESMValCore because then [Dask dashboard](#) remains available after ESMValCore has finished running.

Advice on choosing performant configurations

The threads within a single worker can access the same memory locations, so they may freely pass around chunks, while communicating a chunk between workers is done by copying it, so this is (a bit) slower. Therefore it is beneficial for performance to have multiple threads per worker. However, due to limitations in the CPython implementation (known as the Global Interpreter Lock or GIL), only a single thread in a worker can execute Python code (this limitation does not apply to compiled code called by Python code, e.g. `numpy`), therefore the best performing configurations will typically not use much more than 10 threads per worker.

Due to limitations of the NetCDF library (it is not thread-safe), only one of the threads in a worker can read or write to a NetCDF file at a time. Therefore, it may be beneficial to use fewer threads per worker if the computation is very simple and the runtime is determined by the speed with which the data can be read from and/or written to disk.

Custom Dask threaded scheduler configuration

The Dask threaded scheduler can be a good choice for recipes using a small amount of data or when running a recipe where not all preprocessor functions are lazy yet (see [Issue #674](#) for the current status).

To avoid running out of memory, it is important to set the number of workers (threads) used by Dask to run its computations to a reasonable number. By default, the number of CPU cores in the machine will be used, but this may be too many on shared machines or laptops with a large number of CPU cores compared to the amount of memory they have available.

Typically, Dask requires about 2 GiB of RAM per worker, but this may be more depending on the computation.

To set the number of workers used by the Dask threaded scheduler, use the following configuration:

```
dask:
  use: local_threaded # This can be omitted
  profiles:
    local_threaded:
      num_workers: 4
```

Default options

By default, the following Dask configuration is used:

```
dask:
  use: local_threaded # use the `local_threaded` profile defined below
  profiles:
    local_threaded:
      scheduler: threads
    local_distributed:
      cluster:
        type: distributed.LocalCluster
  debug:
    scheduler: synchronous
```

All available options

Option	Description	Type	Default value
profile	Different Dask profiles that can be selected via the use option. Each profile has a name (<code>dict</code> keys) and corresponding options (<code>dict</code> values). See <i>Options for Dask profiles</i> for details.	<code>dict</code>	See <i>Default options</i>
use	Dask profile that is used; must be defined in the option profiles.	<code>str</code>	<code>local_threaded</code>

Options for Dask profiles

Option	Description	Type	Default value
<code>cluster</code>	Keyword arguments to initialize a Dask distributed cluster. Needs the option <code>type</code> , which specifies the class of the cluster. The remaining options are passed as keyword arguments to initialize that class. Cannot be used in combination with <code>scheduler_address</code> .	<code>dict</code>	If omitted, use externally managed cluster if <code>scheduler_address</code> is given or a <i>Dask threaded scheduler</i> otherwise.
<code>scheduler_address</code>	Scheduler address of an externally managed cluster. Will be passed to <code>distributed.Client</code> . Cannot be used in combination with <code>cluster</code> .	<code>str</code>	If omitted, use a Dask distributed cluster if <code>cluster</code> is given or a <i>Dask threaded scheduler</i> otherwise.
All other options	Passed as keyword arguments to <code>dask.config.set()</code> .	Any	No defaults.

1.2.6 Logging configuration

Configure what information is logged and how it is presented in the logging section.

Note

Not all logging configuration is available here yet, see [Issue #2596](#).

Configuration file example:

```
logging:
  log_progress_interval: 10s
```

will log progress of Dask computations every 10 seconds instead of showing a progress bar.

Command line example:

```
esmvaltool run --logging='{"log_progress_interval": "1m"}' recipe_example.yml
```

will log progress of Dask computations every minute instead of showing a progress bar.

Available options:

Option	Description	Type	Default value
<code>log_progress_interval</code>	When running computations with Dask, log progress every <code>log_progress_interval</code> instead of showing a progress bar. The value can be specified in the format accepted by <code>dask.utils.parse_timedelta()</code> . A negative value disables any progress reporting. A progress bar is only shown if <code>max_parallel_tasks: 1</code> .	<code>str</code> or <code>float</code>	0

1.2.7 Project-specific configuration

Configure project-specific settings in the `projects` section.

Top-level keys in this section are projects, e.g., CMIP6, CORDEX, or obs4MIPs.

Example:

```
projects:
  CMIP6:
    ... # project-specific options
```

The following project-specific options are available:

Option	Description	Type	Default value
<code>cmor_table</code>	<i>CMOR tables</i> are used to define the variables that ESMValCore can work with. Refer to <i>CMOR table configuration</i> for available options.	dict	{"type": "esmvalcore.cmor.table.NoInfo"}
<code>data</code>	Data sources are used to find input data and have to be configured before running the tool. Refer to <i>Data sources</i> for details.	dict	{}
<code>extra_facets</code>	Extra key-value pairs (" <i>facets</i> ") added to datasets in addition to the facets defined in the recipe. Refer to <i>Extra Facets</i> for details.	dict	Refer to <i>Default extra facets</i> .
<code>preprocessor</code>	A template defining the filenames to use for <i>preprocessed data</i> when running a <i>recipe</i> . Refer to <i>Preprocessor output filenames</i> for details.	str	Refer to <i>Preprocessor output filenames</i> .

CMOR table configuration

CMOR tables are used to define the variables that ESMValCore can work with.

Default values are provided in `defaults/cmor_tables.yml`, for example:

Listing 1: First few lines of `defaults/cmor_tables.yml`

```
# CMOR table configuration.
projects:
  # Projects hosted on ESGF.
  CMIP7:
    cmor_table:
      type: esmvalcore.cmor.table.CMIP6Info
      paths:
        - cmip7/tables
        - cmip6-custom
  CMIP6:
    cmor_table:
      type: esmvalcore.cmor.table.CMIP6Info
      paths:
        - cmip6/Tables
        - cmip6-custom
  CMIP5:
    cmor_table:
      type: esmvalcore.cmor.table.CMIP5Info
```

(continues on next page)

(continued from previous page)

paths:

- cmip5/Tables
- cmip5-custom

The `type` parameter defines which class is used to read the CMOR tables for a project, it should be a subclass of `esmvalcore.cmor.table.InfoBase`. The other parameters are passed as keyword arguments to the class when it is created. See `esmvalcore.cmor.table` for a description of the built-in classes for reading CMOR tables and their parameters.

Most users will not need to change the CMOR table configuration. However, if you have variables you would like to use which are not in the standard tables, it is recommended that you start from the default configuration and extend the list of `paths` with the directory where your custom CMOR tables are stored.

If you have data that is not described in a CMOR table at all and do not need interoperability with typical climate model data, you can use the `esmvalcore.cmor.table.NoInfo` class to indicate that no CMOR table is available. In that case you will need to provide all necessary facets for loading and saving the data in the `recipe` or `Dataset`, and `CMOR checks` will be skipped.

Warning

While it is possible to work with datasets that are not described in a CMOR table, the `preprocessor functions` and `diagnostics` have been designed to work with CMORized data and may not work as expected with non-CMORized data.

Our ambition is that `preprocessor functions` support data that follows the `CF Conventions`.

Data sources

The `data` section defines sources of input data. The easiest way to get started with these is to copy one of the example configuration files and tailor it to your needs.

To list the available example configuration files, run the command:

```
esmvaltool config list
```

To use one of the example configuration files, copy it to your configuration directory by running the command:

```
esmvaltool config copy data-intake-esgf.yml
```

where `data-intake-esgf.yml` needs to be replaced by the name of the example configuration you would like to use. The format of the configuration file is described in `esmvalcore.io`.

There are three modules available as part of ESMValCore that provide data sources:

- `esmvalcore.io.intake_esgf`: Use the `intake-esgf` library to load data that is available from ESGF.
- `esmvalcore.io.local`: Use `glob` patterns to find files on a filesystem.
- `esmvalcore.io.esgf`: Use the legacy `esgf-pyclient` library to find and download data from ESGF.

Adding a custom data source is relatively easy and is explained in `esmvalcore.io.protocol`.

There are various use cases and we provide example configurations for each of them below.

Personal computer

On a personal computer, the recommended setup can be obtained by running the commands:

```
esmvaltool config copy data-intake-esgf.yml
esmvaltool config copy data-local-esmvaltool.yml
```

This will use the `esmvalcore.io.intake_esgf` module to access data that is available through ESGF and use `esmvalcore.io.local` to find observational and reanalysis datasets that have been CMORized with ESMValTool (OBS6 and OBS projects for CMIP6- and CMIP5-style CMORization respectively) or are supported in their *native format* through the `native6` project.

Warning

It is important to [configure intake-esgf](#) for your system before using it. Make sure to set `local_cache` to a path where it can store downloaded files, and if (some) ESGF data is already available on your system, point `esg_dataroot` to it. If you are missing certain search results, you may want to choose a different index node for searching the ESGF.

HPC system

On HPC systems, data is often stored in large shared filesystems. We have several example configurations for popular HPC systems. To list the available example files, run the command:

```
esmvaltool config list data-hpc
```

If you are using one of the supported HPC systems, for example Jasmin, you can copy the example configuration file by running the command:

```
esmvaltool config copy data-hpc-badc.yml
```

and you should be good to go. If your HPC system is not supported yet, you can copy one of the other example configuration files, e.g. `data-hpc-dkrz.yml` and tailor it for your system.

Warning

It is important to [configure intake-esgf](#) for your system before using it. Make sure to set `local_cache` to a path where it can store downloaded files, and if (some) ESGF data is already available on your system, point `esg_dataroot` to it. If you are missing certain search results, you may want to choose a different index node for searching the ESGF.

Note

Deduplicating data found via `esmvalcore.io.intake_esgf` data sources and the `esmvalcore.io.local` data sources has not yet been implemented. Therefore it is recommended not to use the configuration option `search_data: complete` when using both data sources for the same project. The `search_data: quick` option can be safely used.

Climate model data in its native format

For each of the climate models that are supported in their native format as described in *Supported native models*, an example configuration file is available. To list the available example files, run the command:

```
esmvaltool config list data-native
```

Filter Iris load warnings

It is possible to ignore specific warnings when loading data with Iris. This is particularly useful for native datasets which do not follow the CMOR standard by default and consequently produce a lot of warnings when handled by Iris. This can be configured using the `ignore_warnings` argument to `esmvalcore.io.local.LocalDataSource`.

Here is an example on how to ignore specific warnings when loading data from the EMAC model in its native format:

```
# Read data from the EMAC model in its native format.
projects:
  EMAC:
    data:
      emac:
        type: esmvalcore.io.local.LocalDataSource
        rootpath: ~/climate_data
        dirname_template: "{exp}/{channel}"
        filename_template: "{exp}*{channel}{postproc_flag}.nc"
        ignore_warnings:
          - message: "Ignored formula of unrecognised type: .*"
            module: iris
          - message: "Ignoring formula terms variable .* referenced by data variable .*"
            module: iris
          - message: "Missing CF-netCDF formula term variable .*, referenced by netCDF"
            module: iris
          - message: "NetCDF variable .* contains unknown cell method .*"
            module: iris
```

The keyword arguments specified in the list items are directly passed to `warnings.filterwarnings()` in addition to `action=ignore`.

Extra Facets

It can be useful to automatically add extra *facets* to variables or datasets without explicitly specifying them in the recipe. These facets can be used for *finding data* or for providing extra information to the functions that *fix data* before passing it on to the preprocessor.

To support this, we provide the **extra facets** facilities. Facets are the key-value pairs described in *Recipe section: datasets*. Extra facets allows for the addition of more details per project, dataset, MIP table, and variable name.

Format of the extra facets

Extra facets are configured in the `extra_facets` section of the project-specific configuration. They are specified in nested dictionaries with the following levels:

1. Dataset name
2. MIP table

3. Variable short name

Example:

```
projects:
  CMIP6:
    extra_facets:
      CanESM5: # dataset name
      Amon: # MIP table
      tas: # variable short name
      a_new_key: a_new_value # extra facets
```

The three top levels under `extra_facets` (dataset name, MIP table, and variable short name) can contain [Unix shell-style wildcards](#). The special characters used in shell-style wildcards are:

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

where `seq` can either be a sequence of characters or just a bunch of characters, for example `[A-C]` matches the characters A, B, and C, while `[AC]` matches the characters A and C.

Examples:

```
projects:
  CMIP6:
    extra_facets:
      CanESM5: # dataset name
      "*" : # MIP table
      "*" : # variable short name
      a_new_key: a_new_value # extra facets
```

Here, the extra facet `a_new_key: a_new_value` will be added to any `CMIP6` data from model `CanESM5`.

If keys are duplicated, later keys will take precedence over earlier keys:

```
projects:
  CMIP6:
    extra_facets:
      CanESM5:
        "*" :
        "*" :
        shared_key: with_wildcard
        unique_key_1: test
      Amon:
        tas:
          shared_key: without_wildcard
          unique_key_2: test
```

Here, the following extra facets will be added to a dataset with project `CMIP6`, name `CanESM5`, MIP table `Amon`, and variable short name `tas`:

```
unique_key_1: test
shared_key: without_wildcard # takes value from later entry
unique_key_2: test
```

Default extra facets

Default extra facets are specified in `extra_facets_*.yaml` files located in [this](#) directory.

Preprocessor output filenames

The filename to use for saving *preprocessed data* when running a *recipe* is configured using `preprocessor_filename_template`, similar to the filename template in `esmvalcore.io.local.LocalDataSource`.

Default values are provided in `defaults/preprocessor_filename_template.yaml`, for example:

Listing 2: First few lines of `defaults/preprocessor_filename_template.yaml`

```
# Templates for the filenames used to write preprocessor output.
projects:
  # ESGF projects.
  CMIP3:
    preprocessor_filename_template: "{project}_{institute}_{dataset}_{mip}_{exp}_
    ↪{ensemble}_{short_name}"
  CMIP5:
    preprocessor_filename_template: "{project}_{dataset}_{mip}_{exp}_{ensemble}_{short_
    ↪name}"
  CMIP6:
    preprocessor_filename_template: "{project}_{dataset}_{mip}_{exp}_{ensemble}_{short_
    ↪name}_{grid}"
  CMIP7:
    preprocessor_filename_template: "{short_name}_{branding_suffix}_{frequency}_{region}_
    ↪{grid}_{dataset}_{exp}_{ensemble}"
  CORDEX:
    preprocessor_filename_template: "{project}_{institute}_{dataset}_{rcm_version}_
    ↪{driver}_{domain}_{mip}_{exp}_{ensemble}_{short_name}"
  obs4MIPs:
    preprocessor_filename_template: "{project}_{dataset}_{short_name}"
  ana4MIPs:
    preprocessor_filename_template: "{project}_{mip}_{type}_{dataset}_{short_name}"
```

The facet names from the template are replaced with the facet values from the recipe to create a filename. The extension `.nc` (and if applicable, a start and end time) will automatically be appended to the filename.

If no `preprocessor_filename_template` is configured for a project, the facets describing the dataset in the recipe, as stored in `esmvalcore.dataset.Dataset.minimal_facets`, are used.

1.2.8 ESGF configuration

The `esmvaltool run` command can automatically download the files required to run a recipe from ESGF for the projects CMIP3, CMIP5, CMIP6, CORDEX, and obs4MIPs.

Refer to *Data sources* for instructions on how to set this up. This section describes additional configuration options for the `esmvalcore.io.esgf` module, which is based on the legacy `esgf-pyclient` library. Most users will not need this.

Note

When running a recipe that uses many or large datasets on a machine that does not have any data available locally, the amount of data that will be downloaded can be in the range of a few hundred gigabyte to a few terrabyte. See [Obtaining input data](#) for advice on getting access to machines with large datasets already available.

A log message will be displayed with the total amount of data that will be downloaded before starting the download. If you see that this is more than you would like to download, stop the tool by pressing the Ctrl and C keys on your keyboard simultaneously several times, edit the recipe so it contains fewer datasets and try again.

Configuration file

An optional configuration file can be created for configuring how the `esmvalcore.io.esgf.ESGFDataSource` uses `esgf-pyclient` to find and download data. The name of this file is `~/.esmvaltool/esgf-pyclient.yml`.

Search

Any arguments to `pyesgf.search.connection.SearchConnection` can be provided in the section `search_connection`, for example:

```
search_connection:
  expire_after: 2592000 # the number of seconds in a month
```

to keep cached search results for a month.

The default settings are:

```
search_connection:
  urls:
    - 'https://esgf-node.ornl.gov/esgf-1-5-bridge'
    - 'https://esgf.ceda.ac.uk/esg-search'
    - 'https://esgf-data.dkrz.de/esg-search'
    - 'https://esgf-node.ipsl.upmc.fr/esg-search'
    - 'https://esgf-dn1.nsc.liu.se/esg-search'
    - 'https://esgf.nci.org.au/esg-search'
    - 'https://esgf.nccs.nasa.gov/esg-search'
    - 'https://esgdata.gfdl.noaa.gov/esg-search'
  distrib: true
  timeout: 120 # seconds
  cache: '~/.esmvaltool/cache/pyesgf-search-results'
  expire_after: 86400 # cache expires after 1 day
```

Note that by default the tool will try searching the [ESGF index nodes](#) in the order provided in the configuration file and use the first one that is online. Some ESGF index nodes may return search results faster than others, so you may be able to speed up the search for files by experimenting with placing different index nodes at the top of the list.

Warning

ESGF is currently transitioning to new server technology and all of the above indices are expected to go offline except the first one.

Issues with <https://esgf-node.ornl.gov/esgf-1-5-bridge> can be reported [here](#).

If you experience errors while searching, it sometimes helps to delete the cached results.

Download statistics

The tool will maintain statistics of how fast data can be downloaded from what host in the file `~/esmvaltool/cache/esgf-hosts.yml` and automatically select hosts that are faster. There is no need to manually edit this file, though it can be useful to delete it if you move your computer to a location that is very different from the place where you previously downloaded data. An entry in the file might look like this:

```
esgf2.dkrz.de:
  duration (s): 8
  error: false
  size (bytes): 69067460
  speed (MB/s): 7.9
```

The tool only uses the duration and size to determine the download speed, the speed shown in the file is not used. If error is set to true, the most recent download request to that host failed and the tool will automatically try this host only as a last resort.

1.2.9 Developer configuration file

Deprecated since version 2.14.0: The developer configuration file is deprecated and will no longer be supported in v2.16.0. Please use the *project-specific configuration* instead.

See the v2.13.0 documentation for previous usage of the developer configuration file.

Warning

Make sure that **no** `config-developer.yml` file is saved in the ESMValCore configuration directories (see *Overview* for details), as it does not contain configuration options that are valid in the new configuration system.

Upgrade instructions for finding files

The `input_dir`, `input_file`, and `ignore_warnings` settings have been replaced by `esmvalcore.io.local.LocalDataSource`, which can be configured via *data sources*.

Example 1: A `config-developer.yml` file specifying a directory structure for CMIP6 data:

```
CMIP6:
  input_dir:
    ESGF: "{project}/{activity}/{institute}/{dataset}/{exp}/{ensemble}/{mip}/{short_name}
    ↔/{grid}/{version}"
  input_file: "{short_name}_{mip}_{dataset}_{exp}_{ensemble}_{grid}*.nc"
```

and associated `rootpath` and `drs` settings:

```
rootpath:
  CMIP6: ~/climate_data
drs:
  CMIP6: ESGF
```

would translate to the following new configuration:

```
projects:
  CMIP6:
    data:
      local:
```

(continues on next page)

(continued from previous page)

```

type: esmvalcore.io.local.LocalDataSource
rootpath: ~/climate_data
dirname_template: "{project}/{activity}/{institute}/{dataset}/{exp}/{ensemble}/
↪{mip}/{short_name}/{grid}/{version}"
filename_template: "{short_name}_{mip}_{dataset}_{exp}_{ensemble}_{grid}*.nc"

```

Upgrade instructions for naming preprocessor output files

The `output_file` setting has been replaced by the `preprocessor_filename_template` settings described in *Pre-processor output filenames*.

Example 1: A config-developer.yml file specifying preprocessor output filenames for CMIP6 data:

```

CMIP6:
  output_file: "{project}_{dataset}_{mip}_{exp}_{ensemble}_{short_name}_{grid}"

```

would translate to the following new configuration:

```

projects:
  CMIP6:
    preprocessor_filename_template: "{project}_{dataset}_{mip}_{exp}_{ensemble}_{short_
↪name}_{grid}"

```

Upgrade instructions for using custom CMOR tables

The CMOR tables can now be configured via *CMOR table configuration*. The following mapping applies:

- `cmor_type` has been replaced by `type`
- `cmor_strict` has been replaced by `strict`
- `cmor_path` has been replaced by `paths`
- `cmor_default_table_prefix` is no longer needed.

Because it is now possible to configure multiple paths to directories containing CMOR tables per project, the `custom` project for specifying additional custom CMOR tables is no longer needed nor supported.

Example 1: A config-developer.yml file specifying different CMIP6 CMOR tables than the default ones, augmented by the default custom CMOR tables:

```

CMIP6:
  cmor_path: /path/to/cmip6-cmor-tables
  cmor_strict: true
  cmor_type: CMIP6

```

would translate to the following new configuration:

```

projects:
  CMIP6:
    cmor_table:
      type: esmvalcore.cmor.table.CMIP6Info
      strict: true
      paths:
        - /path/to/cmip6-cmor-tables
        - cmip6-custom

```

where the `cmip6-custom` relative path refers to `esmvalcore/cmor/tables/cmip6-custom` and `type` refers to `esmvalcore.cmor.table.CMIP6Info`.

Example 2: A `config-developer.yml` file specifying additional custom CMOR tables:

```
CMIP6:
  cmor_path: cmip6
  cmor_strict: true
  cmor_type: CMIP6
custom:
  cmor_path: /path/to/custom-cmip5-style-tables
```

would translate to the following new configuration:

```
projects:
  CMIP6:
    cmor_table:
      type: esmvalcore.cmor.table.CMIP6Info
      strict: true
      paths:
        - cmip6
        - cmip6-custom
        - /path/to/custom-cmip6-style-tables
```

where the relative paths `cmip6` and `cmip6-custom` refer to `esmvalcore/cmor/tables/cmip6` and `esmvalcore/cmor/tables/cmip6-custom` respectively and the directory `/path/to/custom-cmip6-style-tables` contains the additional custom CMOR tables in CMIP6 format. A script to translate custom CMIP5 format tables to CMIP6 format tables is available [here](#). Note that it is no longer possible to mix CMIP6-style tables with custom CMIP5-style tables for the same project.

1.2.10 References configuration file

The `esmvaltool/config-references.yml` file contains the list of ESMValTool diagnostic and recipe authors, references and projects. Each author, project and reference referred to in the documentation section of a recipe needs to be in this file in the relevant section.

For instance, the recipe `recipe_ocean_example.yml` file contains the following documentation section:

```
documentation:
  authors:
    - demo_le

  maintainer:
    - demo_le

  references:
    - demora2018gmd

  projects:
    - ukesm
```

These four items here are named people, references and projects listed in the `config-references.yml` file.

1.3 Input data

1.3.1 Overview

Data discovery and retrieval is the first step in any evaluation process; ESMValCore uses a *semi-automated* data finding mechanism with inputs from both the configuration and the recipe file: this means that the user will have to provide the tool with a set of parameters related to the data needed and once these parameters have been provided, the tool will automatically find the right data. We will detail below the data finding and retrieval process and the input the user needs to specify, giving examples on how to use the data finding routine under different scenarios.

1.3.2 Data types

CMIP data

CMIP data is widely available via the Earth System Grid Federation (ESGF) and is accessible to users either via automatic download by `esmvaltool` or through the ESGF data nodes hosted by large computing facilities (like CEDA-Jasmin, DKRZ, etc). This data adheres to, among other standards, the DRS and Controlled Vocabulary standard for naming files and structured paths; the DRS ensures that files and paths to them are named according to a standardized convention. Examples of this convention, also used by ESMValCore for file discovery and data retrieval, include:

- CMIP6 file: `{variable_short_name}_{mip}_{dataset_name}_{experiment}_{ensemble}_{grid}_{start-date}-{end-date}.nc`
- CMIP5 file: `{variable_short_name}_{mip}_{dataset_name}_{experiment}_{ensemble}_{start-date}-{end-date}.nc`
- OBS file: `{project}_{dataset_name}_{type}_{version}_{mip}_{short_name}_{start-date}-{end-date}.nc`

Similar standards exist for the standard paths (input directories); for the ESGF data nodes, these paths differ slightly, for example:

- CMIP6 path for BADC: `ROOT-BADC/{institute}/{dataset_name}/{experiment}/{ensemble}/{mip}/{variable_short_name}/{grid}`;
- CMIP6 path for ETHZ: `ROOT-ETHZ/{experiment}/{mip}/{variable_short_name}/{dataset_name}/{ensemble}/{grid}`

From the ESMValCore user perspective the number of data input parameters is optimized to allow for ease of use. We detail this procedure in the next section.

Observational data

Part of observational data is retrieved in the same manner as CMIP data, for example using the OBS root path set to:

```
OBS: /gws/ssde/j25a/esmeval/obsdata-v2
```

and the dataset:

```
- {dataset: ERA-Interim, project: OBS6, type: reanaly, version: 1, start_year: ↵
↵2014, end_year: 2015, tier: 3}
```

in `recipe.yml` in `datasets` or `additional_datasets`, the rules set in *CMOR-DRS* are used again and the file will be automatically found:

```
/gws/ssde/j25a/esmeval/obsdata-v2/Tier3/ERA-Interim/OBS_ERA-Interim_reanaly_1_Amon_ta_
↵201401-201412.nc
```

Observational datasets CMORized by ESMValTool are organized in Tiers depending on their level of public availability.

Datasets in native format

Some datasets are supported in their native format (i.e., the data is not formatted according to a CMIP data request) through the `native6` project (mostly native reanalysis/observational datasets) or through a dedicated project, e.g., `ICON` (mostly native models). A detailed description of how to include new native datasets is given [here](#).

Supported native reanalysis/observational datasets

The following native reanalysis/observational datasets are supported under the `native6` project. To use these datasets, put the files containing the data in the directory that you have *configured* for the `rootpath` of the `native6` project, in a subdirectory called `Tier{tier}/{dataset}/{version}/{frequency}/{short_name}`, assuming you are using the default DRS for `native6` as defined in the file:

Listing 3: Contents of `data-local-esmvaltool.yml`

```

projects:
  # Data that can be read in its native format by ESMValCore.
  native6:
    data:
      local:
        type: esmvalcore.io.local.LocalDataSource
        rootpath: ~/climate_data
        dirname_template: "Tier{tier}/{dataset}/{version}/{frequency}/{short_name}"
        filename_template: "*.nc"

```

Replace the items in curly braces by the values used in the variable/dataset definition in the *recipe*.

ERA5 (in netCDF format downloaded from the CDS)

ERA5 data can be downloaded from the Copernicus Climate Data Store (CDS) using the convenient tool `era5cli`. For example for monthly data, place the files in the `/Tier3/ERA5/version/mon/pr` subdirectory of your `rootpath` that you have configured for the `native6` project (assuming you are using the default DRS for `native6` described [above](#)).

- Supported variables: `cl`, `clt`, `evspsbl`, `evspsblpot`, `mrro`, `pr`, `prsn`, `ps`, `psl`, `ptype`, `rls`, `rlds`, `rsds`, `rsdt`, `rss`, `uas`, `vas`, `tas`, `tasmx`, `tasmin`, `tdps`, `ts`, `tsn` (E1hr/Amon), `orog` (fx).
- Tier: 3

Note

According to the description of Evapotranspiration and potential Evapotranspiration on the Copernicus page (<https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels-monthly-means?tab=overview>): “The ECMWF Integrated Forecasting System (IFS) convention is that downward fluxes are positive. Therefore, negative values indicate evaporation and positive values indicate condensation.”

In the CMOR table, these fluxes are defined as positive, if they go from the surface into the atmosphere: “Evaporation at surface (also known as evapotranspiration): flux of water into the atmosphere due to conversion of both liquid and solid phases to vapor (from underlying surface and vegetation).” Therefore, the ERA5 (and ERA5-Land) CMORizer switches the signs of `evspsbl` and `evspsblpot` to be compatible with the CMOR standard used e.g. by the CMIP models.

ERA5 (in GRIB format available on DKRZ's Levante or downloaded from the CDS)

ERA5 data in monthly, daily, and hourly resolution is available on Levante in its native GRIB format.

Note

ERA5 data in its native GRIB format can also be downloaded from the [Copernicus Climate Data Store \(CDS\)](#). For example, hourly data on pressure levels is available [here](#). Reading self-downloaded ERA5 data in GRIB format is experimental and likely requires additional setup from the user like setting up the proper directory structure for the input files.

To read these data with ESMValCore, use the data definition for the `native6` project:

Listing 4: Contents of `data-hpc-dkrz-era5-grib.yml`

```
# Read ERA5 data in GRIB format on Levante at DKRZ (see
# https://docs.dkrz.de/doc/dataservices/finding_and_accessing_data/era_data/index.html
# ↪ #pool-data-era5-file-and-directory-names
# for details).
projects:
  native6:
    data:
      dkrz-era5-grib:
        type: esmvalcore.io.local.LocalDataSource
        rootpath: /pool/data/ERA5
        dirname_template: "{family}/{level}/{type}/{tres}/{grib_id}"
        filename_template: "{family}{level}{typeid}_{tres}_*_{grib_id}.grb"
```

To use this configuration, run `esmvaltool config copy data-hpc-dkrz-era5-grib.yml`.

The [naming conventions](#) for input directories and files for native ERA5 data in GRIB format on Levante are

- input directories: `{family}/{level}/{type}/{tres}/{grib_id}`
- input files: `{family}{level}{typeid}_{tres}_*_{grib_id}.grb`

All of these facets have reasonable defaults preconfigured in the corresponding *extra facets* configuration file, which is available here: `extra_facets_native6.yml`. If necessary, these facets can be overwritten in the recipe.

Thus, example dataset entries could look like this:

```
datasets:
- {project: native6, dataset: ERA5, timerange: '2000/2001',
  short_name: tas, mip: Amon}
- {project: native6, dataset: ERA5, timerange: '2000/2001',
  short_name: cl, mip: Amon, tres: 1H, frequency: 1hr}
- {project: native6, dataset: ERA5, timerange: '2000/2001',
  short_name: ta, mip: Amon, type: fc, typeid: '12'}
```

The native ERA5 output in GRIB format is stored on a [reduced Gaussian grid](#). By default, these data are regridded to a regular $0.25^\circ \times 0.25^\circ$ grid as recommended by the ECMWF using bilinear interpolation.

To disable this, you can use the facet `automatic_regrid: false` in the recipe:

datasets:

```
- {project: native6, dataset: ERA5, timerange: '2000/2001',
  short_name: tas, mip: Amon, automatic_regrid: false}
```

- Supported variables: albsn, cl, cli, clt, clw, hur, hus, o3, prw, ps, psl, rainmxrat27, sftlf, snd, snowmxrat27, ta, tas, tdps, toz, ts, ua, uas, va, vas, wap, zg.

MSWEP

- Supported variables: pr
- Supported frequencies: mon, day, 3hr.
- Tier: 3

For example for monthly data, place the files in the `/Tier3/MSWEP/version/mon/pr` subdirectory of your rootpath that you have configured for the `native6` project (assuming you are using the default DRS for `native6` described *above*).

Note

For monthly data (V220), the data must be postfixed with the date, i.e. rename `global_monthly_050deg.nc` to `global_monthly_050deg_197901-201710.nc`

For more info: <http://www.gloh2o.org/>

Data for the version V220 can be downloaded from: https://hydrology.princeton.edu/data/hylkeb/MSWEP_V220/.

Supported native models

The following models are natively supported by ESMValCore. In contrast to the native observational datasets listed above, they use dedicated projects instead of the project `native6`.

CESM

ESMValCore is able to read native `CESM` model output.

Warning

The support for native `CESM` output is still experimental. Currently, only one variable (`tas`) is fully supported. Other 2D variables might be supported by specifying appropriate facets in the recipe or extra facets files (see text below). 3D variables (data that uses a vertical dimension) are not supported, yet.

The default naming conventions for input directories and files for `CESM` are

- **input directories: 3 different types supported:**
 - / (run directory)
 - {case}/{gcomp}/hist (short-term archiving)
 - {case}/{gcomp}/proc/{tdir}/{tperiod} (post-processed data)
- input files: {case}.{scomp}.{type}.{string}*nc

as configured in:

Listing 5: Contents of data-native-cesm.yml

```
# Read data from the CESM model in its native format.
projects:
  CESM:
    data:
      run: &cesm
      type: esmvalcore.io.local.LocalDataSource
      rootpath: ~/climate_data
      dirname_template: "" # run directory
      filename_template: "{case}.{scomp}.{type}.{string}*nc"
      short-term-archive:
        <<: *cesm
      dirname_template: "{case}/{gcomp}/hist" # short-term archiving
      postprocessed:
        <<: *cesm
      dirname_template: "{case}/{gcomp}/proc/{tdir}/{tperiod}" # postprocessed data
```

To use this configuration, run `esmvaltool config copy data-native-cesm.yml` and adapt the `rootpath` to your system. More information about CESM naming conventions are given [here](#).

Note

The `{string}` entry in the input file names above does not only correspond to the (optional) `$string` entry for [CESM model output files](#), but can also be used to read [post-processed files](#). In the latter case, `{string}` corresponds to the combination `$$STRING.$TSTRING`.

Thus, example dataset entries could look like this:

```
datasets:
- {project: CESM, dataset: CESM2, case: f.e21.FHIST_BGC.f09_f09_mg17.CMIP6-AMIP.001,
  ↪ type: h0, mip: Amon, short_name: tas, start_year: 2000, end_year: 2014}
- {project: CESM, dataset: CESM2, case: f.e21.F1850_BGC.f09_f09_mg17.CFMIP-hadsst-
  ↪ piForcing.001, type: h0, gcomp: atm, scomp: cam, mip: Amon, short_name: tas, start_
  ↪ year: 2000, end_year: 2014}
```

Variable-specific defaults for the facet `gcomp` and `scomp` are given as extra facets (see next paragraph) for some variables, but these can be overwritten in the recipe.

Similar to any other fix, the CESM fix allows the use of *extra facets*. The configuration file `extra_facets_cesm.yml` contains the defaults. Currently, this file only contains default facets for a single variable (`tas`); for other variables, these entries need to be defined in the recipe. Supported keys for extra facets are:

Key	Description	Default value if not specified
<code>gcomp</code>	Generic component-model name	No default (needs to be specified as extra facets or in recipe if default DRS is used)
<code>raw_r</code>	Variable name of the variable in the raw input file	CMOR variable name of the corresponding variable
<code>raw_u</code>	Units of the variable in the raw input file	If specified, the value given by the <code>units</code> attribute in the raw input file; otherwise unknown
<code>scomp</code>	Specific component-model name	No default (needs to be specified as extra facets or in recipe if default DRS is used)
<code>strir</code>	Short string which is used to further identify the history file type (corresponds to <code>\$string</code> or <code>\$SSTRING</code> . <code>\$TSTRING</code> in the CESM file name conventions; see note above)	' ' (empty string)
<code>tdir</code>	Entry to distinguish time averages from time series from diagnostic plot sets (only used for post-processed data)	' ' (empty string)
<code>tperi</code>	Time period over which the data was processed (only used for post-processed data)	' ' (empty string)

EMAC

ESMValCore is able to read native EMAC model output.

The default naming conventions for input directories and files for EMAC are

- input directories: `{exp}/{channel}`
- input files: `{exp}*{channel}{postproc_flag}.nc`

as configured in:

Listing 6: Contents of `data-native-emas.yml`

```
# Read data from the EMAC model in its native format.
projects:
  EMAC:
    data:
      emacs:
        type: esmvalcore.io.local.LocalDataSource
        rootpath: ~/climate_data
        dirname_template: "{exp}/{channel}"
        filename_template: "{exp}*{channel}{postproc_flag}.nc"
        ignore_warnings:
          - message: "Ignored formula of unrecognised type: .*"
            module: iris
          - message: "Ignoring formula terms variable .* referenced by data variable .*"
            module: iris
          - message: "Missing CF-netCDF formula term variable .*, referenced by netCDF"
            module: iris
          - message: "NetCDF variable .* contains unknown cell method .*"
            module: iris
```

To use this configuration, run `esmvaltool config copy data-native-emas.yml` and adapt the rootpath to your system.

Thus, example dataset entries could look like this:

```
datasets:
- {project: EMAC, dataset: EMAC, exp: historical, mip: Amon, short_name: tas, start_
↪year: 2000, end_year: 2014}
- {project: EMAC, dataset: EMAC, exp: historical, mip: Omon, short_name: tos, postproc_
↪flag: "-p-mm", start_year: 2000, end_year: 2014}
- {project: EMAC, dataset: EMAC, exp: historical, mip: Amon, short_name: ta, raw_name:
↪tm1_p39_cav, start_year: 2000, end_year: 2014}
```

Please note the duplication of the name EMAC in project and dataset, which is necessary to comply with ESMValCore's data finding and CMORizing functionalities. A variable-specific default for the facet channel is given in the extra facets (see next paragraph) for many variables, but this can be overwritten in the recipe.

Similar to any other fix, the EMAC fix allows the use of *extra facets*. The configuration file `extra_facets_emas.yml` contains the defaults. For some variables, extra facets are necessary; otherwise ESMValCore cannot read them properly. Supported keys for extra facets are:

Key	Description	Default value if not specified
channel	Channel in which the desired variable is stored	No default (needs to be specified as extra facets or in recipe if default DRS is used)
postproc_fl	Postprocessing flag of the data	' ' (empty string)
raw_name	Variable name of the variable in the raw input file	CMOR variable name of the corresponding variable
raw_units	Units of the variable in the raw input file	If specified, the value given by the <code>units</code> attribute in the raw input file; otherwise <code>unknown</code>
reset_time_	Boolean if time bounds are deleted, and automatically recalculated by iris	False

Note

`raw_name` can be given as `str` or `list`. The latter is used to support multiple different variables names in the input file. In this case, the prioritization is given by the order of the list; if possible, use the first entry, if this is not present, use the second, etc. This is particularly useful for files in which regular averages (`*_ave`) or conditional averages (`*_cav`) exist.

For 3D variables defined on pressure levels, only the pressure levels defined by the CMOR table (e.g., for *Amon's ta*: `tm1_p19_cav` and `tm1_p19_ave`) are given as default extra facets. If other pressure levels are desired, e.g., `tm1_p39_cav`, this has to be explicitly specified in the recipe using `raw_name: tm1_p39_cav` or `raw_name: [tm1_p19_cav, tm1_p39_cav]`.

ICON

ESMValCore is able to read native **ICON** model output.

The default naming conventions for input directories and files for **ICON** are

- input directories: `{exp}`, `{exp}/outdata`, or `{exp}/output`
- input files: `{exp}_{var_type}*.nc`

as configured in:

Listing 7: Contents of data-native-icon.yml

```
# Read data from the ICON model in its native format.
projects:
  ICON:
    data:
      icon: &icon
      type: esmvalcore.io.local.LocalDataSource
      rootpath: ~/climate_data
      dirname_template: "{exp}"
      filename_template: "{exp}_{var_type}*.nc"
      ignore_warnings:
        - message: "Failed to create 'height' dimension coordinate: The 'height'
↳DimCoord bounds array must be strictly monotonic."
          module: iris
      icon-outdata:
        <<: *icon
        dirname_template: "{exp}/outdata"
      icon-output:
        <<: *icon
        dirname_template: "{exp}/output"
```

To use this configuration, run `esmvaltool config copy data-native-icon.yml` and adapt the `rootpath` to your system.

Currently, two different versions of ICON are supported:

1. ICON-A, which is based on ECHAM physics (deprecated): select via `dataset: ICON`.
2. ICON-XPP, which is based on NWP physics (preferred; model code can be downloaded from [DKRZ's GitLab](#)): select via `dataset: ICON-XPP`.

Thus, example dataset entries could look like this:

```
datasets:
- {project: ICON, dataset: ICON, exp: icon-2.6.1_atm_amip_R2B5_r1i1p1f1,
  mip: Amon, short_name: tas, timerange: 20010101/20020101}
- {project: ICON, dataset: ICON-XPP, exp: historical, mip: Amon,
  short_name: ta, timerange: 20010101/20020101}
```

A variable-specific default for the facet `var_type` is given in the extra facets (see below) for many variables, but this can be overwritten in the recipe, for example:

```
datasets:
- {project: ICON, dataset: ICON-XPP, exp: historical, mip: Amon,
  short_name: ta, var_type: atm_dyn_3d_ml, timerange: 20010101/20020101}
```

This is necessary if your ICON output is structured in one variable per file. For example, if your output is stored in files called `<exp>_<variable_name>_atm_2d_ml_YYYYMMDDThhmmss.nc`, use `var_type: <variable_name>_atm_2d_ml` in the recipe for this variable.

Usually, ICON reports aggregated values at the end of the corresponding time output intervals. For example, for monthly output, ICON reports the month February as “1 March”. Thus, by default, ESMValCore shifts all time points back by 1/2 of the output time interval so that the new time point corresponds to the center of the interval. This can be disabled by using `shift_time: false` in the recipe or the extra facets (see below). For point measurements (identified by `cell_methods = "time: point"`), this is always disabled.

Warning

To get all desired time points, do **not** use `start_year` and `end_year` in the recipe, but rather `timerange` with at least 8 digits. For example, to get data for the years 2000 and 2001, use `timerange: 20000101/20020101` instead of `timerange: 2000/2001` or `start_year: 2000, end_year: 2001`. See [Time ranges](#) for more information on the `timerange` option.

Usually, ESMValCore will need the corresponding ICON grid file of your simulation to work properly (examples: setting latitude/longitude coordinates if these are not yet present, UGRIDization [see below], etc.). This grid file can either be specified as absolute or relative (to the *configuration option* `auxiliary_data_dir`) path with the facet `horizontal_grid` in the recipe or as extra facet (see below), or retrieved automatically from the `grid_file_uri` attribute of the input files. In the latter case, ESMValCore first searches the input directories specified for ICON for a grid file with that name, and if that was not successful, tries to download the file and cache it. The cached file is valid for 7 days.

ESMValCore can automatically make native ICON data UGRID-compliant when loading the data. The UGRID conventions provide a standardized format to store data on unstructured grids, which is required by many software packages or tools to work correctly and specifically by Iris to interpret the grid as a *mesh*. An example is the horizontal regridding of native ICON data to a regular grid. While the *built-in regridding schemes* `linear` and `nearest` can handle unstructured grids (i.e., not UGRID-compliant) and meshes (i.e., UGRID-compliant), the `area_weighted` scheme requires the input data in UGRID format. This automatic UGRIDization is enabled by default, but can be switched off with the facet `ugrid: false` in the recipe or as extra facet (see below). This is useful for diagnostics that act on the native ICON grid and do not support input data in UGRID format (yet).

For 3D ICON variables, ESMValCore tries to add the pressure level information and/or altitude information to the pre-processed output files. If the names of these variables differ from the default values, the facets `pfull_var`, `phalf_var`, `zg_var`, and `zghalf_var` can be specified in the recipe or as extra facets. If neither of these variables are available in the input files, it is possible to specify the location of files that include the corresponding altitude information with the facets `zg_file` and/or `zghalf_file` in the recipe or as extra facets. The paths to these files can be specified absolute or relative (to the *configuration option* `auxiliary_data_dir`).

Hint

To use the `extract_levels()` preprocessor on native ICON data, you need to specify the name of the vertical coordinate (e.g., `coordinate: air_pressure`) since native ICON output usually provides a 3D air pressure field instead of a simple 1D vertical coordinate. This also works if your files only contain altitude information (in this case, the US standard atmosphere is used to convert between altitude and pressure levels; see [Vertical interpolation](#) for details). Example:

```
preprocessors:
  extract_500hPa_level_from_icon:
    extract_levels:
      levels: 50000
      scheme: linear
      coordinate: air_pressure
```

Similar to any other fix, the ICON fix allows the use of *extra facets*. The configuration file `extra_facets_icon.yml` contains the defaults. For some variables, extra facets are necessary; otherwise ESMValCore cannot read them properly. Supported keys for extra facets are:

Key	Description	Default value if not specified
<code>horizont</code>	Absolute or relative (to <code>auxiliary_data_dir</code>) path to the ICON grid file	If not given, use file attribute <code>grid_file_uri</code> to retrieve ICON grid file (see details above)
<code>lat_var</code>	Variable name of the latitude coordinate in the raw input file/grid file	<code>clat</code>
<code>lon_var</code>	Variable name of the longitude coordinate in the raw input file/grid file	<code>clon</code>
<code>pfull_va</code>	Variable name of the pressure at full levels in the raw input file	<code>pfull</code> (dataset: ICON) or <code>pres</code> (dataset: ICON-XPP)
<code>phalf_va</code>	Variable name of the pressure at half levels in the raw input file	<code>phalf</code>
<code>raw_name</code>	Variable name of the variable in the raw input file	CMOR variable name of the corresponding variable
<code>raw_unit</code>	Units of the variable in the raw input file	If specified, the value given by the <code>units</code> attribute in the raw input file; otherwise <code>unknown</code>
<code>shift_ti</code>	Shift time points back by 1/2 of the corresponding output time interval	<code>True</code>
<code>ugrid</code>	Automatic UGRIDization of the input data	<code>True</code>
<code>var_type</code>	Variable type of the variable in the raw input file	No default (needs to be specified as extra facets or in recipe if default DRS is used)
<code>zg_file</code>	Absolute or relative (to <code>auxiliary_data_dir</code>) path to the the input file that contains the geometric height at full levels	If possible, use geometric height at full levels provided by the raw input file
<code>zg_var</code>	Variable name of the geometric height at full levels in the raw input file	<code>zg</code>
<code>zghalf_f</code>	Absolute or relative (to <code>auxiliary_data_dir</code>) path to the the input file that contains the geometric height at half levels	If possible, use geometric height at half levels provided by the raw input file
<code>zghalf_v</code>	Variable name of the geometric height at half levels in the raw input file	<code>zghalf</code>

Hint

In order to read cell area files (`areacella` and `areacello`), one additional manual step is necessary: Copy the ICON grid file (you can find a download link in the global attribute `grid_file_uri` of your ICON data) to your ICON input directory and change its name in such a way that only the grid file is found when the cell area variables are required. Make sure that this file is not found when other variables are loaded.

For example, you could use a new `var_type`, e.g., `horizontalgrid` for this file. Thus, an ICON grid file located in `2.6.1_atm_amip_R2B5_r1i1p1f1/2.6.1_atm_amip_R2B5_r1i1p1f1_horizontalgrid.nc` can be found using `var_type: horizontalgrid` in the recipe (assuming the default naming conventions listed above). Make sure that no other variable uses this `var_type`.

If you want to use the `area_statistics()` preprocessor on *regridded* ICON data, make sure to **not** use the cell area files by using the `skip: true` syntax in the recipe as described in *Supplementary variables (ancillary variables and cell measures)*, e.g.,

datasets:

- `{project: ICON, dataset: ICON, exp: amip, supplementary_variables: [{short_name: areacella, skip: true}]}`

IPSL-CM6

Both output formats (i.e. the Output and the Analyse / Time series formats) are supported, and should be configured in recipes as e.g.:

```
datasets:
- {simulation: CM61-LR-hist-03.1950, exp: piControl, out: Analyse, freq: TS_M0,
  account: p86caub, status: PROD, dataset: IPSL-CM6, project: IPSLCM,
  root: /thredds/tgcc/store}
- {simulation: CM61-LR-hist-03.1950, exp: historical, out: Output, freq: M0,
  account: p86caub, status: PROD, dataset: IPSL-CM6, project: IPSLCM,
  root: /thredds/tgcc/store}
```

and data can be found by running `esmvaltool config copy data-native-ipslicm.yml` and adapting the rootpath to your system.

The Output format is an example of a case where variables are grouped in multi-variable files, which name cannot be computed directly from datasets attributes alone but requires the usage *Extra Facets*. The configuration file `extra_facets_ipslicm.yml` contains the default extra facets. These multi-variable files must also undergo some data selection.

ACCESS-ESM

ESMValTool can read native ACCESS-ESM model output.

Warning

This is the first version of ACCESS-ESM CMORizer for ESMValCore. Currently, Supported variables: pr, ps, psl, rlds, tas, ta, va, ua, zg, hus, clt, rsus, rlus.

The default naming conventions for input directories and files for ACCESS output are

- input directories: `{institute}/{sub_dataset}/{exp}/{modeling_realm}/netCDF`
- input files: `{sub_dataset}.{special_attr}*.nc`

as configured in:

Listing 8: Contents of `data-native-access.yml`

```
# Read data from the ACCESS model in its native format.
projects:
ACCESS:
  data:
    access-sub-dataset:
      type: esmvalcore.io.local.LocalDataSource
      rootpath: ~/climate_data
      dirname_template: "{dataset}/{sub_dataset}/{exp}/{modeling_realm}/netCDF"
      filename_template: "{sub_dataset}.{freq_attribute}*.nc"
    access-ocean:
      type: esmvalcore.io.local.LocalDataSource
      rootpath: ~/climate_data
      dirname_template: "{dataset}/{sub_dataset}/{exp}/{modeling_realm}/netCDF"
      filename_template: "ocean_{freq_attribute}.nc-*
```

To use this configuration, run `esmvaltool config copy data-native-access.yml` and adapt the rootpath to your system.

Thus, example dataset entries could look like this:

```
dataset:
- {project: ACCESS, mip: Amon, dataset:ACCESS_ESM1_5, sub_dataset: HI-CN-05,
  exp: history, modeling_realm: atm, special_attr: pa, start_year: 1986, end_year:
↪1986}
```

Similar to any other fix, the ACCESS-ESM fix allows the use of *extra facets*. The configuration file `extra_facets_access.yml` contains the defaults. For some variables, extra facets are necessary; otherwise ESMValCore cannot read them properly. Supported keys for extra facets are:

Key	Description	Default value if not specified
<code>raw_name</code>	Variable name of the variable in the raw input file	CMOR variable name of the corresponding variable
<code>modeling_realm</code>	Realm attribute includes <i>atm</i> , <i>ice</i> , and <i>oce</i>	No default (needs to be specified as extra facets or in recipe if default DRS is used)
<code>freq_attr</code>	A special attribute in the filename <i>ACCESS-ESM</i> raw data, related to the frequency of raw data	No default
<code>sub_dataset</code>	Part of the ACCESS-ESM raw dataset root, needs to be specified if you want to use the cmoriser	No default
<code>ocean_grid</code>	Path to load the grid data for ACCESS ocean variables	No default

1.3.3 Data retrieval

Please go to [Data sources](#) for instructions and background on how to configure data retrieval.

1.3.4 Data loading

Data loading is done using the data load functionality of *iris*; we will not go into too much detail about this since we can point the user to the specific functionality [here](#) but we will underline that the initial loading is done by adhering to the CF Conventions that *iris* operates by as well (see [CF Conventions Document](#) and the search page for CF standard names).

1.3.5 Data concatenation from multiple sources

Oftentimes data retrieving results in assembling a continuous data stream from multiple files or even, multiple experiments. The internal mechanism through which the assembly is done is via cube concatenation. One peculiarity of *iris* concatenation (see [iris cube concatenation](#)) is that it doesn't allow for concatenating time-overlapping cubes; this case is rather frequent with data from models overlapping in time, and is accounted for by a function that performs a flexible concatenation between two cubes, depending on the particular setup:

- cubes overlap in time: resulting cube is made up of the overlapping data plus left and right hand sides on each side of the overlapping data; note that in the case of the cubes coming from different experiments the resulting concatenated cube will have composite data made up from multiple experiments: assume [cube1: exp1, cube2: exp2] and cube1 starts before cube2, and cube2 finishes after cube1, then the concatenated cube will be made up of cube2: exp2 plus the section of cube1: exp1 that contains data not provided in cube2: exp2;
- cubes don't overlap in time: data from the two cubes is bolted together;

Note that two cube concatenation is the base operation of an iterative process of reducing multiple cubes from multiple data segments via cube concatenation ie if there is no time-overlapping data, the cubes concatenation is performed in

one step.

1.3.6 Use of extra facets in the datafinder

Extra facets are a mechanism to provide additional information for certain kinds of data. The general approach is described in *Extra Facets*. Here, we describe how they can be used to locate data files within the datafinder framework. This is useful to build paths for directory structures and file names that require more information than what is provided in the recipe. A common application is the location of variables in multi-variable files as often found in climate models' native output formats.

Another use case is files that use different names for variables in their file name than for the netCDF4 variable name.

To apply the extra facets for this purpose, simply use the corresponding tag in the applicable `filename_template` or `dirname_template` in `esmvalcore.io.local.LocalDataSource`.

For example, given the extra facets

```
projects:
  native6:
    extra_facets:
      ERA5:
        Amon:
          tas:
            source_var_name: t2m
```

a corresponding entry in the configuration file could look like:

Listing 9: Contents of `data-local-esmvaltool.yml`

```
# Data that can be read in its native format by ESMValCore.
native6:
  data:
    local:
      type: esmvalcore.io.local.LocalDataSource
      rootpath: ~/climate_data
      dirname_template: "Tier{tier}/{dataset}/{version}/{frequency}/{short_name}"
      filename_template: "*.nc"
```

The same replacement mechanism can be employed everywhere where tags can be used, particularly in `dirname_template` and `filename_template` in `esmvalcore.io.local.LocalDataSource`, and in `preprocessor_filename_template` under *Project-specific configuration*.

1.4 Running

The ESMValCore package provides the `esmvaltool` command line tool, which can be used to run a *recipe*.

To list the available commands, run

```
esmvaltool --help
```

It is also possible to get help on specific commands, e.g.

```
esmvaltool run --help
```

will display the help message with all options for the `run` command.

To run a recipe, call `esmvaltool run` with the path to the desired recipe:

```
esmvaltool run recipe_example.yml
```

The `esmvaltool run recipe_example.yml` command will first look if `recipe_example.yml` is the path to an existing file. If this is the case, it will run that recipe. If you have [ESMValTool installed](#), it will look if the name matches one of the recipes in your ESMValTool installation directory, in the subdirectory `recipes` and run that.

Note

There is no `recipe_example.yml` shipped with either ESMValCore or ESMValTool. If you would like to try out the command above, replace `recipe_example.yml` with the path to an existing recipe, e.g. `examples/recipe_python.yml` if you have the ESMValTool package installed.

To work with installed recipes, the ESMValTool package provides the `esmvaltool recipes` command, see [Available diagnostics and metrics](#).

By default, ESMValTool searches for *configuration files* in `~/config/esmvaltool`. If you'd like to use a custom location, you can specify this via the `--config_dir` command line argument:

```
esmvaltool run --config_dir /path/to/custom_config recipe_example.yml
```

It is also possible to explicitly set configuration options with command line arguments:

```
esmvaltool run --argument_name argument_value recipe_example.yml
```

To control the strictness of the CMOR checker and the checks during concatenation on auxiliary coordinates, supplementary variables, and derived coordinates, use the flag `--check_level`:

```
esmvaltool run --check_level=relaxed recipe_example.yml
```

Possible values are:

- *ignore*: all errors will be reported as warnings. Concatenation will be performed without checks.
- *relaxed*: only fail if there are critical errors. Concatenation will be performed without checks.
- *default*: fail if there are any errors.
- *strict*: fail if there are any warnings.

To reuse pre-processed files from a previous run of the same recipe, you can use

```
esmvaltool run recipe_example.yml --resume_from ~/esmvaltool_output/recipe_python_
↳20210930_123907
```

Multiple directories can be specified for reuse, make sure to quote them:

```
esmvaltool run recipe_example.yml --resume_from "~/esmvaltool_output/recipe_python_
↳20210930_101007 ~/esmvaltool_output/recipe_python_20210930_123907"
```

The first preprocessor directory containing the required data will be used.

This feature can be useful when developing new diagnostics, because it avoids the need to re-run the preprocessor. Another potential use case is running the preprocessing part of a recipe on one or more machines that have access to a lot of data and then running the diagnostics on a machine without access to data.

To run only the preprocessor tasks from a recipe, use

```
esmvaltool run recipe_example.yml --remove_preproc_dir=False --run_diagnostic=False
```

Note

Only preprocessing *tasks* that completed successfully can be reused with the `--resume_from` option. Preprocessing tasks that completed successfully, contain a file called *metadata.yml* in their output directory.

To run a reduced version of the recipe, usually for testing purpose you can use

```
esmvaltool run --max_datasets=NDATASETS --max_years=NYEARS recipe_example.yml
```

In this case, the recipe will limit the number of datasets per variable to `NDATASETS` and the total amount of years loaded to `NYEARS`. They can also be used separately. Note that diagnostics may require specific combinations of available data, so use the above two flags at your own risk and for testing purposes only.

To run a recipe, even if some datasets are not available, use

```
esmvaltool run --skip_nonexistent=True recipe_example.yml
```

It is also possible to select only specific diagnostics to be run. To run only one, just specify its name. To provide more than one diagnostic to filter use the syntax `'diag1 diag2/script1'` or `'("diag1", "diag2/script1")'` and pay attention to the quotes.

```
esmvaltool run --diagnostics=diagnostic1 recipe_example.yml
```

Note

ESMValTool command line interface is created using the Fire python package. This package supports the creation of completion scripts for the Bash and Fish shells. Go to <https://google.github.io/python-fire/using-cli/#python-fires-flags> to learn how to set up them.

1.5 Output

ESMValTool automatically generates a new output directory with every run. The location is determined by the `output_dir` *configuration option*, the recipe name, and the date and time, using the the format: `YYYYMMDD_HHMMSS`.

For instance, a typical output location (*session_dir*) would be: `output_directory/recipe_ocean_amoc_20190118_1027/`

This is effectively produced by the combination: `output_dir/recipe_name_YYYYMMDD_HHMMSS/`

This directory will contain 4 further subdirectories:

1. *Diagnostic output (work)*: A place for any diagnostic script results that are not plots, e.g. files in NetCDF format (depends on the diagnostics).
2. *Plots (plots)*: The location for all the plots, split by individual diagnostics and fields.
3. *Run (run)*: This directory includes all log files, a copy of the recipe, a summary of the resource usage, and the *settings.yml* interface files and temporary files created by the diagnostic scripts.
4. *Preprocessed datasets (preproc)*: This directory contains all the preprocessed netcdfs data and the *metadata.yml* interface files. Note that by default this directory will be deleted after each run, because most users will only need the results from the diagnostic scripts.

A summary of the output is produced in the file: `index.html`

1.5.1 Preprocessed datasets

The preprocessed datasets will be stored to the `preproc/` directory. Each variable in each diagnostic will have its own the `metadata.yml` interface files saved in the `preproc` directory.

If the *configuration option* `save_intermediary_cubes` is set to `true`, then the intermediary cubes will also be saved here (default: `false`).

If the *configuration option* `remove_preproc_dir` is set to `true`, then the `preproc` directory will be deleted after the run completes (default: `true`).

1.5.2 Run

The log files in the run directory are automatically generated by ESMValTool and create a record of the output messages produced by ESMValTool and they are saved in the run directory. They can be helpful for debugging or monitoring the job, but also allow a record of the job output to screen after the job has been completed.

The run directory will also contain a copy of the recipe and the `settings.yml` file, described below. The run directory is also where the diagnostics are executed, and may also contain several temporary files while diagnostics are running.

1.5.3 Diagnostic output

The `work/` directory will contain all files that are output at the diagnostic stage. Ie, the model data is preprocessed by ESMValTool and stored in the `preproc/` directory. These files are opened by the diagnostic script, then some processing is applied. Once the diagnostic level processing has been applied, the results should be saved to the work directory.

1.5.4 Plots

The plots directory is where diagnostics save their output figures. These plots are saved in the format requested by the *configuration option* `output_file_type`.

1.5.5 Settings.yml

The `settings.yml` file is automatically generated by ESMValTool. Each diagnostic will produce a unique `settings.yml` file.

The `settings.yml` file passes several global level keys to diagnostic scripts. This includes several flags from the global configuration (such as `log_level`), several paths which are specific to the diagnostic being run (such as `plot_dir` and `run_dir`) and the location on disk of the `metadata.yml` file (described below).

```
input_files: [...]recipe_ocean_bgc_20190118_134855/preproc/diag_timeseries_scalars/mfo/
↳ metadata.yml
log_level: debug
output_file_type: png
plot_dir: [...]recipe_ocean_bgc_20190118_134855/plots/diag_timeseries_scalars/Scalar_
↳ timeseries
profile_diagnostic: false
recipe: recipe_ocean_bgc.yml
run_dir: [...]recipe_ocean_bgc_20190118_134855/run/diag_timeseries_scalars/Scalar_
↳ timeseries
script: Scalar_timeseries
version: 2.0a1
work_dir: [...]recipe_ocean_bgc_20190118_134855/work/diag_timeseries_scalars/Scalar_
↳ timeseries
```

The first item in the settings file will be a list of *Metadata.yml* files. There is a metadata.yml file generated for each field in each diagnostic.

1.5.6 Metadata.yml

The metadata.yml files is automatically generated by ESMValTool. Along with the settings.yml file, it passes all the paths, boolean flags, and additional arguments that your diagnostic needs to know in order to run.

The metadata is loaded from cfg as a dictionary object in python diagnostics.

Here is an example metadata.yml file:

```
?
[...]/recipe_ocean_bgc_20190118_134855/preproc/diag_timeseries_scalars/mfo/CMIP5_
↪HadGEM2-ES_Omon_historical_r1i1p1_T00M_mfo_2002-2004.nc
: cmor_table: CMIP5
dataset: HadGEM2-ES
diagnostic: diag_timeseries_scalars
end_year: 2004
ensemble: r1i1p1
exp: historical
field: T00M
filename: [...]/recipe_ocean_bgc_20190118_134855/preproc/diag_timeseries_scalars/mfo/
↪CMIP5_HadGEM2-ES_Omon_historical_r1i1p1_T00M_mfo_2002-2004.nc
frequency: mon
institute: [INPE, MOHC]
long_name: Sea Water Transport
mip: Omon
modeling_realm: [ocean]
preprocessor: prep_timeseries_scalar
project: CMIP5
recipe_dataset_index: 0
short_name: mfo
standard_name: sea_water_transport_across_line
start_year: 2002
units: kg s-1
variable_group: mfo
```

As you can see, this is effectively a dictionary with several items including data paths, metadata and other information.

There are several tools available in python which are built to read and parse these files. The tools are available in the shared directory in the diagnostics directory.

EXAMPLE NOTEBOOKS

Example notebooks are available in the [notebooks](#) folder and can also be viewed here. These notebooks demonstrate the use of the *Python API*.

2.1 Composing recipes

This notebooks shows how to fill the datasets section in a [recipe](#).

```
import yaml

from esmvalcore.config import CFG
from esmvalcore.dataset import Dataset, datasets_to_recipe
```

Configure ESMValCore so it always searches the ESGF for data

```
CFG["search_esgf"] = "always"
```

Here is a small example recipe, that uses the `datasets_to_recipe` function to convert a list of datasets to a recipe:

```
tas = Dataset(
    short_name="tas",
    mip="Amon",
    project="CMIP6",
    dataset="CanESM5-1",
    ensemble="r1i1p1f1",
    exp="historical",
    grid="gn",
    timerange="2000/2002",
)
tas["diagnostic"] = "diagnostic_name"

pr = tas.copy(short_name="pr")

print(yaml.safe_dump(datasets_to_recipe([tas, pr])))
```

```
datasets:
- dataset: CanESM5-1
diagnostics:
  diagnostic_name:
    variables:
      pr:
```

(continues on next page)

(continued from previous page)

```

ensemble: r1i1p1f1
exp: historical
grid: gn
mip: Amon
project: CMIP6
timerange: 2000/2002
tas:
ensemble: r1i1p1f1
exp: historical
grid: gn
mip: Amon
project: CMIP6
timerange: 2000/2002

```

A more ambitious recipe might want to use all data that is available on ESGF. We can define a dataset template with a facet value of * where any value can be used. This can then be expanded to a list of datasets using the `from_files()` method.

```

dataset_template = Dataset(
    short_name="tas",
    mip="Amon",
    project="CMIP6",
    exp="historical",
    dataset="*",
    institute="*",
    ensemble="*",
    grid="*",
)
datasets = list(dataset_template.from_files())
len(datasets)

```

778

This results in the following recipe:

```

for dataset in datasets:
    dataset.facets["diagnostic"] = "diagnostic_name"
print(yaml.safe_dump(datasets_to_recipe(datasets)))

```

```

datasets:
- dataset: TaiESM1
  ensemble: r(1:2)i1p1f1
  grid: gn
  institute: AS-RCEC
- dataset: AWI-CM-1-1-MR
  ensemble: r(1:5)i1p1f1
  grid: gn
  institute: AWI
- dataset: AWI-ESM-1-1-LR
  ensemble: r1i1p1f1
  grid: gn
  institute: AWI

```

(continues on next page)

(continued from previous page)

```
- dataset: BCC-CSM2-MR
  ensemble: r(1:3)ilp1f1
  grid: gn
  institute: BCC
- dataset: BCC-ESM1
  ensemble: r(1:3)ilp1f1
  grid: gn
  institute: BCC
- dataset: CAMS-CSM1-0
  ensemble: r1ilp1f2
  grid: gn
  institute: CAMS
- dataset: CAMS-CSM1-0
  ensemble: r(1:2)ilp1f1
  grid: gn
  institute: CAMS
- dataset: CAS-ESM2-0
  ensemble: r(1:4)ilp1f1
  grid: gn
  institute: CAS
- dataset: FGOALS-f3-L
  ensemble: r(1:3)ilp1f1
  grid: gr
  institute: CAS
- dataset: FGOALS-g3
  ensemble: r(1:6)ilp1f1
  grid: gn
  institute: CAS
- dataset: IITM-ESM
  ensemble: r1ilp1f1
  grid: gn
  institute: CCCR-IITM
- dataset: CanESM5-1
  ensemble: r(1:20)ilp1f1
  grid: gn
  institute: CCCma
- dataset: CanESM5-1
  ensemble: r(1:25)ilp2f1
  grid: gn
  institute: CCCma
- dataset: CanESM5-1
  ensemble: r22ilp1f1
  grid: gn
  institute: CCCma
- dataset: CanESM5-1
  ensemble: r(24:39)ilp1f1
  grid: gn
  institute: CCCma
- dataset: CanESM5-1
  ensemble: r(41:50)ilp1f1
  grid: gn
  institute: CCCma
```

(continues on next page)

(continued from previous page)

```
- dataset: CanESM5-CanOE
  ensemble: r(1:3)ilp2f1
  grid: gn
  institute: CCCma
- dataset: CanESM5
  ensemble: r(1:25)ilp1f1
  grid: gn
  institute: CCCma
- dataset: CanESM5
  ensemble: r(1:40)ilp2f1
  grid: gn
  institute: CCCma
- dataset: CMCC-CM2-HR4
  ensemble: r1ilp1f1
  grid: gn
  institute: CMCC
- dataset: CMCC-CM2-SR5
  ensemble: r1ilp1f1
  grid: gn
  institute: CMCC
- dataset: CMCC-CM2-SR5
  ensemble: r(2:11)ilp2f1
  grid: gn
  institute: CMCC
- dataset: CMCC-ESM2
  ensemble: r1ilp1f1
  grid: gn
  institute: CMCC
- dataset: CNRM-CM6-1-HR
  ensemble: r1ilp1f2
  grid: gr
  institute: CNRM-CERFACS
- dataset: CNRM-CM6-1
  ensemble: r(1:30)ilp1f2
  grid: gr
  institute: CNRM-CERFACS
- dataset: CNRM-ESM2-1
  ensemble: r(1:11)ilp1f2
  grid: gr
  institute: CNRM-CERFACS
- dataset: ACCESS-CM2
  ensemble: r(1:10)ilp1f1
  grid: gn
  institute: CSIRO-ARCCSS
- dataset: ACCESS-ESM1-5
  ensemble: r(1:40)ilp1f1
  grid: gn
  institute: CSIRO
- dataset: E3SM-1-0
  ensemble: r(1:5)ilp1f1
  grid: gr
  institute: E3SM-Project
```

(continues on next page)

(continued from previous page)

```
- dataset: E3SM-1-1-ECA
  ensemble: r1i1p1f1
  grid: gr
  institute: E3SM-Project
- dataset: E3SM-1-1
  ensemble: r1i1p1f1
  grid: gr
  institute: E3SM-Project
- dataset: E3SM-2-0
  ensemble: r(1:5)i1p1f1
  grid: gr
  institute: E3SM-Project
- dataset: EC-Earth3-AerChem
  ensemble: r1i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3-AerChem
  ensemble: r(3:4)i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3-CC
  ensemble: r1i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3-CC
  ensemble: r4i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3-CC
  ensemble: r(6:13)i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3-Veg-LR
  ensemble: r(1:3)i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3-Veg
  ensemble: r(1:6)i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3-Veg
  ensemble: r10i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3-Veg
  ensemble: r12i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3-Veg
  ensemble: r14i1p1f1
  grid: gr
  institute: EC-Earth-Consortium
```

(continues on next page)

(continued from previous page)

```
- dataset: EC-Earth3
  ensemble: r(1:7)ilp1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3
  ensemble: r(9:25)ilp1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: EC-Earth3
  ensemble: r(101:150)ilp1f1
  grid: gr
  institute: EC-Earth-Consortium
- dataset: FIO-ESM-2-0
  ensemble: r(1:3)ilp1f1
  grid: gn
  institute: FIO-QLNM
- dataset: MPI-ESM-1-2-HAM
  ensemble: r(1:3)ilp1f1
  grid: gn
  institute: HAMMOZ-Consortium
- dataset: INM-CM4-8
  ensemble: r1ilp1f1
  grid: gr1
  institute: INM
- dataset: INM-CM5-0
  ensemble: r(1:10)ilp1f1
  grid: gr1
  institute: INM
- dataset: IPSL-CM5A2-INCA
  ensemble: r1ilp1f1
  grid: gr
  institute: IPSL
- dataset: IPSL-CM6A-LR-INCA
  ensemble: r1ilp1f1
  grid: gr
  institute: IPSL
- dataset: IPSL-CM6A-LR
  ensemble: r(1:33)ilp1f1
  grid: gr
  institute: IPSL
- dataset: KIOST-ESM
  ensemble: r1ilp1f1
  grid: gr1
  institute: KIOST
- dataset: MIROC-ES2H
  ensemble: r1ilp(1:3)f2
  grid: gn
  institute: MIROC
- dataset: MIROC-ES2H
  ensemble: r(1:3)ilp4f2
  grid: gn
  institute: MIROC
```

(continues on next page)

(continued from previous page)

```
- dataset: MIROC-ES2L
  ensemble: r1i1000p1f2
  grid: gn
  institute: MIROC
- dataset: MIROC-ES2L
  ensemble: r(1:30)i1p1f2
  grid: gn
  institute: MIROC
- dataset: MIROC6
  ensemble: r(1:50)i1p1f1
  grid: gn
  institute: MIROC
- dataset: HadGEM3-GC31-LL
  ensemble: r(1:5)i1p1f3
  grid: gn
  institute: MOHC
- dataset: HadGEM3-GC31-MM
  ensemble: r(1:4)i1p1f3
  grid: gn
  institute: MOHC
- dataset: UKESM1-0-LL
  ensemble: r(1:4)i1p1f2
  grid: gn
  institute: MOHC
- dataset: UKESM1-0-LL
  ensemble: r(5:7)i1p1f3
  grid: gn
  institute: MOHC
- dataset: UKESM1-0-LL
  ensemble: r(8:12)i1p1f2
  grid: gn
  institute: MOHC
- dataset: UKESM1-0-LL
  ensemble: r(16:19)i1p1f2
  grid: gn
  institute: MOHC
- dataset: UKESM1-1-LL
  ensemble: r1i1p1f2
  grid: gn
  institute: MOHC
- dataset: ICON-ESM-LR
  ensemble: r(1:5)i1p1f1
  grid: gn
  institute: MPI-M
- dataset: MPI-ESM1-2-HR
  ensemble: r(1:10)i1p1f1
  grid: gn
  institute: MPI-M
- dataset: MPI-ESM1-2-LR
  ensemble: r1i2000p1f1
  grid: gn
  institute: MPI-M
```

(continues on next page)

(continued from previous page)

```
- dataset: MPI-ESM1-2-LR
  ensemble: r(1:30)i1p1f1
  grid: gn
  institute: MPI-M
- dataset: MRI-ESM2-0
  ensemble: r1i2p1f1
  grid: gn
  institute: MRI
- dataset: MRI-ESM2-0
  ensemble: r1i1000p1f1
  grid: gn
  institute: MRI
- dataset: MRI-ESM2-0
  ensemble: r(1:10)i1p1f1
  grid: gn
  institute: MRI
- dataset: GISS-E2-1-G-CC
  ensemble: r1i1p1f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-G
  ensemble: r(1:4)i1p5f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-G
  ensemble: r(1:5)i1p1f3
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-G
  ensemble: r(1:10)i1p1f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-G
  ensemble: r(1:10)i1p3f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-G
  ensemble: r(1:11)i1p1f2
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-G
  ensemble: r(6:10)i1p5f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-G
  ensemble: r(101:102)i1p1f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-H
  ensemble: r(1:5)i1p1f2
  grid: gn
  institute: NASA-GISS
```

(continues on next page)

(continued from previous page)

```
- dataset: GISS-E2-1-H
  ensemble: r(1:5)ilp3f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-H
  ensemble: r(1:5)ilp5f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-1-H
  ensemble: r(1:10)ilp1f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-2-G
  ensemble: r(1:5)ilp3f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-2-G
  ensemble: r(1:6)ilp1f1
  grid: gn
  institute: NASA-GISS
- dataset: GISS-E2-2-H
  ensemble: r(1:5)ilp1f1
  grid: gn
  institute: NASA-GISS
- dataset: CESM2-FV2
  ensemble: r1i2p2f1
  grid: gn
  institute: NCAR
- dataset: CESM2-FV2
  ensemble: r(1:3)ilp1f1
  grid: gn
  institute: NCAR
- dataset: CESM2-WACCM-FV2
  ensemble: r(1:3)ilp1f1
  grid: gn
  institute: NCAR
- dataset: CESM2-WACCM
  ensemble: r(1:3)ilp1f1
  grid: gn
  institute: NCAR
- dataset: CESM2
  ensemble: r(1:11)ilp1f1
  grid: gn
  institute: NCAR
- dataset: NorCPM1
  ensemble: r(1:30)ilp1f1
  grid: gn
  institute: NCC
- dataset: NorESM2-LM
  ensemble: r(1:3)ilp1f1
  grid: gn
  institute: NCC
```

(continues on next page)

(continued from previous page)

```

- dataset: NorESM2-MM
  ensemble: r(1:3)i1p1f1
  grid: gn
  institute: NCC
- dataset: KACE-1-0-G
  ensemble: r(1:3)i1p1f1
  grid: gr
  institute: NIMS-KMA
- dataset: UKESM1-0-LL
  ensemble: r(13:15)i1p1f2
  grid: gn
  institute: NIMS-KMA
- dataset: GFDL-CM4
  ensemble: r1i1p1f1
  grid: gr1
  institute: NOAA-GFDL
- dataset: GFDL-ESM4
  ensemble: r(1:3)i1p1f1
  grid: gr1
  institute: NOAA-GFDL
- dataset: NESM3
  ensemble: r(1:5)i1p1f1
  grid: gn
  institute: NUIST
- dataset: SAM0-UNICON
  ensemble: r1i1p1f1
  grid: gn
  institute: SNU
- dataset: CIESM
  ensemble: r(1:3)i1p1f1
  grid: gr
  institute: THU
- dataset: MCM-UA-1-0
  ensemble: r1i1p1f(1:2)
  grid: gn
  institute: UA
diagnostics:
  diagnostic_name:
    variables:
      tas:
        exp: historical
        mip: Amon
        project: CMIP6

```

2.2 Discovering data

This notebook shows how to find out what data is available locally as well as on ESGF. It also shows how to download the data from ESGF.

```

from esmvalcore.config import CFG
from esmvalcore.dataset import Dataset

```

(continues on next page)

(continued from previous page)

```
from esmvalcore.esgf import download
```

Configure ESMValCore so it always searches the ESGF for data

```
CFG["search_esgf"] = "always"
```

We define a dataset template to search for all CMIP6 datasets that provide surface air temperature (tas) on a monthly resolution for the historical experiment. Note that ESMValCore uses its own names for the facets for a more uniform naming across different CMIP phases and other projects. The mapping to the facet names used on ESGF can be found in `esmvalcore.esgf.facets.FACETS`.

```
dataset_template = Dataset(
    short_name="tas",
    mip="Amon",
    project="CMIP6",
    exp="historical",
    dataset="",
    institute="",
    ensemble="",
    grid="",
)
```

Next, we use the `Dataset.from_files` method to build a list of datasets from the available files. This may take a while as searching the ESGF for many files is a bit slow. Because the search results are cached for a [configurable duration](#), subsequent searches will be faster.

```
datasets = list(dataset_template.from_files())
print(f"Found {len(datasets)} datasets, showing the first 10:")
datasets[:10]
```

```
Found 778 datasets, showing the first 10:
```

```
[Dataset:
{'dataset': 'TaiESM1',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r1i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'AS-RCEC'},
Dataset:
{'dataset': 'TaiESM1',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r2i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'AS-RCEC'},
Dataset:
{'dataset': 'AWI-CM-1-1-MR',
```

(continues on next page)

(continued from previous page)

```
'project': 'CMIP6',
'mip': 'Amon',
'short_name': 'tas',
'ensemble': 'r1i1p1f1',
'exp': 'historical',
'grid': 'gn',
'institute': 'AWI'},
Dataset:
{'dataset': 'AWI-CM-1-1-MR',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r2i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'AWI'},
Dataset:
{'dataset': 'AWI-CM-1-1-MR',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r3i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'AWI'},
Dataset:
{'dataset': 'AWI-CM-1-1-MR',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r4i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'AWI'},
Dataset:
{'dataset': 'AWI-CM-1-1-MR',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r5i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'AWI'},
Dataset:
{'dataset': 'AWI-ESM-1-1-LR',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r1i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'AWI'},
```

(continues on next page)

(continued from previous page)

```

Dataset:
{'dataset': 'BCC-CSM2-MR',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r1i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'BCC'},
Dataset:
{'dataset': 'BCC-CSM2-MR',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r2i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'BCC'}]

```

Let's look at the first dataset in more detail. We can print the facets describing the dataset:

```
dataset = datasets[0]
dataset
```

```

Dataset:
{'dataset': 'TaiESM1',
 'project': 'CMIP6',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r1i1p1f1',
 'exp': 'historical',
 'grid': 'gn',
 'institute': 'AS-RCEC'}

```

and see what files are available:

```
dataset.files
```

```
[ESGFFile:CMIP6/CMIP/AS-RCEC/TaiESM1/historical/r1i1p1f1/Amon/tas/gn/v20200623/tas_Amon_
↪TaiESM1_historical_r1i1p1f1_gn_185001-201412.nc on hosts ['esgf-data1.llnl.gov', 'esgf.
↪ceda.ac.uk', 'esgf.rcec.sinica.edu.tw', 'esgf3.dkrz.de', 'esgf-data04.diasjp.net',
↪'esgf.nci.org.au', 'esgf3.dkrz.de']]
```

A single file can be downloaded using its download method:

```
dataset.files[0].download(CFG["download_dir"])
```

```
LocalFile('~/.climate_data/CMIP6/CMIP/AS-RCEC/TaiESM1/historical/r1i1p1f1/Amon/tas/gn/
↪v20200623/tas_Amon_TaiESM1_historical_r1i1p1f1_gn_185001-201412.nc')
```

For downloading many files, the `esmvalcore.esgf.download` function is recommended because it will download the files in parallel. The ESMValCore will try to guess the fastest host and download from there. If it is not available for some reason, it will automatically fall back to the next host.

```
download(dataset.files, CFG["download_dir"])
```

2.3 Loading, processing, and visualizing data

This notebook shows how to load a dataset, use the preprocessor functions, and visualize the result. In this notebook we will plot the annual mean temperature from 1850 till 2100 from one model.

```
%matplotlib inline

import iris.quickplot
import matplotlib.pyplot as plt

from esmvalcore.config import CFG
from esmvalcore.dataset import Dataset
from esmvalcore.esgf import ESGFFile, download
from esmvalcore.preprocessor import annual_statistics, area_statistics
```

Configure ESMValCore so it searches the ESGF for data

```
CFG["search_esgf"] = "when_missing"
```

Define the dataset we are going to use. In this case surface air temperature (tas).

```
tas = Dataset(
    short_name="tas",
    mip="Amon",
    project="CMIP5",
    dataset="MPI-ESM-MR",
    ensemble="r1i1p1",
    exp="historical",
    timerange="1850/2000",
)
tas
```

```
Dataset:
{'dataset': 'MPI-ESM-MR',
 'project': 'CMIP5',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r1i1p1',
 'exp': 'historical',
 'timerange': '1850/2000'}
```

In order to compute the area average later on, we add the cell areas (areacella) as a supplementary dataset. This will append a new dataset to the list of supplementary datasets. Its facets are copied from the `tas` dataset and updated with the provided facets:

```
tas.add_supplementary(short_name="areacella", mip="fx", ensemble="r0i0p0")
tas.supplementaries
```

```
[Dataset:
 {'dataset': 'MPI-ESM-MR',
```

(continues on next page)

(continued from previous page)

```
'project': 'CMIP5',
'mip': 'fx',
'short_name': 'areacella',
'ensemble': 'r0i0p0',
'exp': 'historical',
'timerange': '1850/2000']}]
```

ESMValCore can automatically add extra facets based on the `project`, `mip`, `short_name`, and `dataset`. These extra facets are automatically added and used when searching for input files.

```
tas.augment_facets()
tas
```

```
Dataset:
{'dataset': 'MPI-ESM-MR',
 'project': 'CMIP5',
 'mip': 'Amon',
 'short_name': 'tas',
 'ensemble': 'r1i1p1',
 'exp': 'historical',
 'frequency': 'mon',
 'institute': ['MPI-M'],
 'long_name': 'Near-Surface Air Temperature',
 'modeling_realm': ['atmos'],
 'original_short_name': 'tas',
 'product': ['output1', 'output2'],
 'standard_name': 'air_temperature',
 'timerange': '1850/2000',
 'units': 'K'}
supplementaries:
{'dataset': 'MPI-ESM-MR',
 'project': 'CMIP5',
 'mip': 'fx',
 'short_name': 'areacella',
 'ensemble': 'r0i0p0',
 'exp': 'historical',
 'frequency': 'fx',
 'institute': ['MPI-M'],
 'long_name': 'Atmosphere Grid-Cell Area',
 'modeling_realm': ['atmos', 'land'],
 'original_short_name': 'areacella',
 'product': ['output1', 'output2'],
 'standard_name': 'cell_area',
 'units': 'm2'}
session: 'session-686367c0-001c-4864-839d-c20887cf7415_20230301_160531'
```

Use the `find_files` method to find the files corresponding to the dataset.

```
tas.find_files()
print(tas.files)
for supplementary_ds in tas.supplementaries:
    print(supplementary_ds.files)
```

```
[ESGFFile:cmip5/output1/MPI-M/MPI-ESM-MR/historical/mon/atmos/Amon/r1i1p1/v20120503/tas_
↪Amon_MPI-ESM-MR_historical_r1i1p1_185001-200512.nc on hosts ['aims3.llnl.gov', 'esgf.
↪ceda.ac.uk', 'esgf.nci.org.au', 'esgf1.dkrz.de']]
[ESGFFile:cmip5/output1/MPI-M/MPI-ESM-MR/historical/fx/atmos/fx/r0i0p0/v20120503/
↪areacella_fx_MPI-ESM-MR_historical_r0i0p0.nc on hosts ['aims3.llnl.gov', 'esgf.ceda.ac.
↪uk', 'esgf.nci.org.au', 'esgf1.dkrz.de']]
```

If the files are not available locally, ESMValCore can download them for us.

```
files = list(tas.files)
for supplementary_ds in tas.supplementaries:
    files.extend(supplementary_ds.files)
files = [file for file in files if isinstance(file, ESGFFile)]
download(files, CFG["download_dir"])
tas.find_files()
print(tas.files)
for supplementary_ds in tas.supplementaries:
    print(supplementary_ds.files)
```

```
[LocalFile('~/.climate_data/cmip5/output1/MPI-M/MPI-ESM-MR/historical/mon/atmos/Amon/
↪r1i1p1/v20120503/tas_Amon_MPI-ESM-MR_historical_r1i1p1_185001-200512.nc')]
[LocalFile('~/.climate_data/cmip5/output1/MPI-M/MPI-ESM-MR/historical/fx/atmos/fx/r0i0p0/
↪v20120503/areacella_fx_MPI-ESM-MR_historical_r0i0p0.nc')]
```

The data in the files can be loaded into an `iris.cube.Cube`. ESMValCore will automatically check for and fix problems with the data formatting and attach the cell area.

```
cube = tas.load()
cube
```

```
<iris 'Cube' of air_temperature / (K) (time: 1812; latitude: 96; longitude: 192)>
```

```
cell_area = cube.cell_measures()[0]
cell_area
```

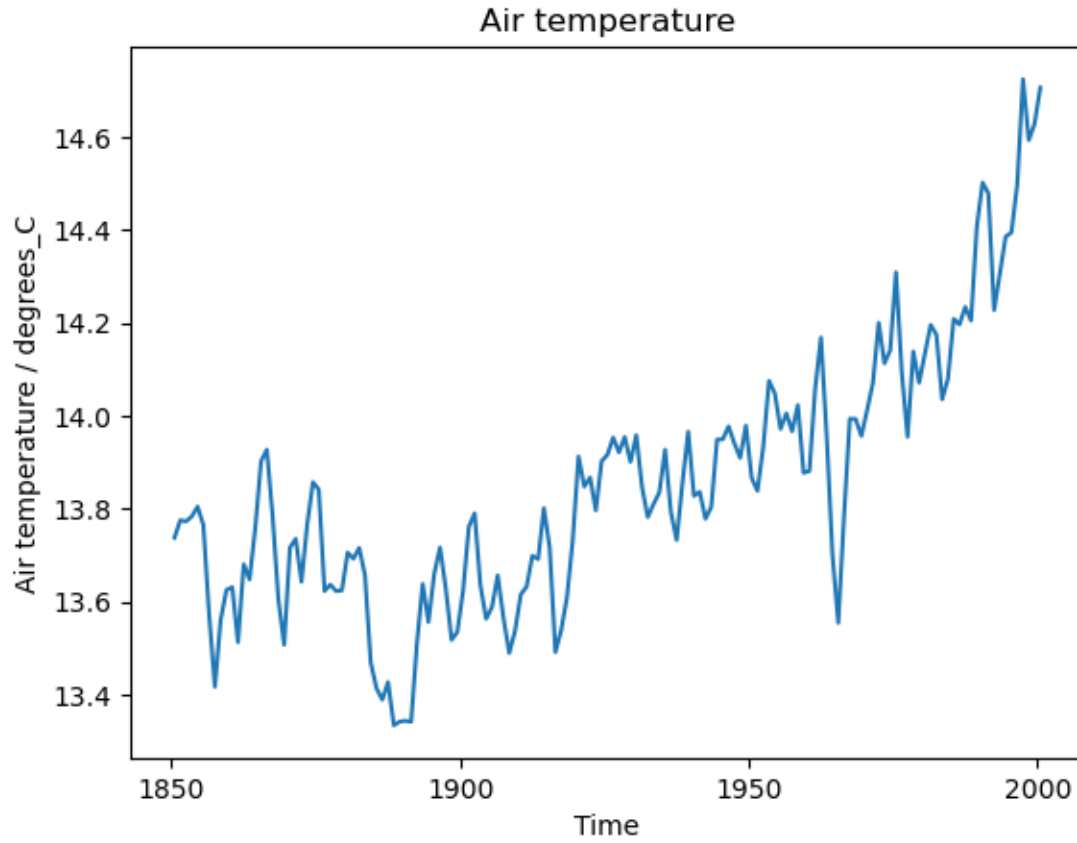
```
<CellMeasure: cell_area / (m2) <lazy> shape(96, 192)>
```

This code shows how to use some `esmvalcore.preprocessor` functions to compute the global annual mean temperature in degrees Celsius:

```
cube = area_statistics(cube, operator="mean")
cube = annual_statistics(cube, operator="mean")
cube.convert_units("degrees_C")
```

The `iris.quickplot` module has useful functions for quickly plotting the results:

```
iris.quickplot.plot(cube)
plt.show()
```



THE RECIPE FORMAT

3.1 Overview

The recipe is the main control file of ESMValTool. It is the only required argument for the `esmvaltool` command line program. Recipes contain the data and data analysis information and instructions needed to run the diagnostic(s), as well as specific diagnostic-related instructions.

Broadly, recipes contain a general section summarizing the provenance and functionality of the diagnostics, the datasets which need to be run, the preprocessors that need to be applied, and the diagnostics which need to be run over the preprocessed data. This information is provided to ESMValTool in four main recipe sections: *Documentation*, *Datasets*, *Preprocessors*, and *Diagnostics*, respectively.

3.1.1 Recipe section: documentation

The documentation section includes:

- The recipe's author's user name (`authors`, matching the definitions in the *References configuration file*)
- The recipe's maintainer's user name (`maintainer`, matching the definitions in the *References configuration file*)
- The title of the recipe (`title`)
- A description of the recipe (`description`, written in Markdown format)
- A list of scientific references (`references`, matching the definitions in the *References configuration file*)
- the project or projects associated with the recipe (`projects`, matching the definitions in the *References configuration file*)

For example, the documentation section of `recipes/recipe_ocean_amoc.yml` is the following:

```
documentation:
  title: Atlantic Meridional Overturning Circulation (AMOC) and the drake passage current
  description: |
    Recipe to produce time series figures of the derived variable, the
    Atlantic meridional overturning circulation (AMOC).
    This recipe also produces transect figures of the stream functions for
    the years 2001-2004.

  authors:
    - demo_le

  maintainer:
    - demo_le
```

(continues on next page)

(continued from previous page)

references:

- demora2018gmd

projects:

- ukesm

Note

Note that all authors, projects, and references mentioned in the description section of the recipe need to be included in the (locally installed copy of the) file `esmvaltool/config-references.yml`, see *References configuration file*. The author name uses the format: `surname_name`. For instance, John Doe would be: `doe_john`. This information can be omitted by new users whose name is not yet included in `config-references.yml`.

3.1.2 Recipe section: datasets

The `datasets` section includes dictionaries that, via *facets*, define standardized data specifications:

- dataset name (key `dataset`, value e.g. `MPI-ESM-LR` or `UKESM1-0-LL`).
- project (key `project`, value `CMIP5` or `CMIP6` for CMIP data, `OBS` for observational data, `ana4MIPs` for ana4MIPs data, `obs4MIPs` for obs4MIPs data, `ICON` for ICON data).
- experiment (key `exp`, value e.g. `historical`, `amip`, `piControl`, `rcp85`).
- mip (for CMIP data, key `mip`, value e.g. `Amon`, `Omon`, `LImon`).
- ensemble member (key `ensemble`, value e.g. `r1i1p1`, `r1i1p1f1`).
- sub-experiment id (key `sub_experiment`, value e.g. `s2000`, `s(2000:2002)`, for DCP data only).
- time range (e.g. key-value `start_year: 1982, end_year: 1990`). Please note that `yaml` interprets numbers with a leading `0` as octal numbers, so we recommend to avoid them. For example, use `128` to specify the year 128 instead of `0128`. Alternatively, the time range can be specified in *ISO 8601 format*, for both dates and periods. In addition, wildcards (`*`) are accepted, which allow the selection of the first available year for each individual dataset (when used as a starting point) or the last available year (when used as an ending point). The starting point and end point must be separated with `/` (e.g. key-value `timerange: '1982/1990'`). More examples are given *here*.
- model grid (native grid `grid: gn` or regridded grid `grid: gr`, for CMIP6 data only).

For example, a `datasets` section could be:

```
datasets:
- {dataset: CanESM2, project: CMIP5, exp: historical, ensemble: r1i1p1, start_year: 2001, end_year: 2004}
- {dataset: UKESM1-0-LL, project: CMIP6, exp: historical, ensemble: r1i1p1f2, start_year: 2001, end_year: 2004, grid: gn}
- {dataset: ACCESS-CM2, project: CMIP6, exp: historical, ensemble: r1i1p1f2, timerange: 'P5Y/*', grid: gn}
- {dataset: EC-EARTH3, alias: custom_alias, project: CMIP6, exp: historical, ensemble: r1i1p1f1, start_year: 2001, end_year: 2004, grid: gn}
- {dataset: CMCC-CM2-SR5, project: CMIP6, exp: historical, ensemble: r1i1p1f1, timerange: '2001/P10Y', grid: gn}
- {dataset: HadGEM3-GC31-MM, project: CMIP6, exp: dcppA-hindcast, ensemble: r1i1p1f1, sub_experiment: s2000, grid: gn, start_year: 2000, end_year: 2002}
```

(continues on next page)

(continued from previous page)

```
- {dataset: BCC-CSM2-MR, project: CMIP6, exp: dcppA-hindcast, ensemble: r1i1p1f1, sub_
↪ experiment: s2000, grid: gn, timerange: '*'}
```

Automatically populating a recipe with all available datasets

It is possible to use `glob` patterns or wildcards for certain facet values, to make it easy to find all available datasets locally and/or on ESGF. Note that `project` cannot be a wildcard.

The facet values for local files are retrieved from the directory tree where the directories represent the facets values. Reading facet values from file names is not yet supported. See *CMIP data* for more information on this kind of file organization.

When (some) files are available locally, the tool will not automatically look for more files on ESGF. To populate a recipe with all available datasets from ESGF, the *configuration option* `search_data` should be set to `complete` and an *ESGF data source* needs to be configured.

For more control over which datasets are selected, it is recommended to use a Python script or *Jupyter notebook* to compose the recipe. See the *Composing recipes* notebook for an example. This is particularly useful when specific relations are required between datasets, e.g. when a dataset needs to be available for multiple variables or experiments.

An example recipe that will use all CMIP6 datasets and all ensemble members which have a 'historical' experiment could look like this:

```
datasets:
- project: CMIP6
  exp: historical
  dataset: '*'
  institute: '*'
  ensemble: '*'
  grid: '*'
```

After running the recipe, a copy specifying exactly which datasets were used is available in the output directory in the run subdirectory. The filename of this recipe will end with `_filled.yml`.

For the `timerange` facet, special syntax is available. See *Time ranges* for more information.

If populating a recipe using wildcards does not work, this is because there were either no files found that match those facets, or the facets could not be read from the directory name or ESGF.

Defining supplementary variables (ancillary variables and cell measures)

It is common practice to store ancillary variables (e.g. land/sea/ice masks) and cell measures (e.g. cell area, cell volume) in separate datasets that are described by slightly different facets. In ESMValCore, we call ancillary variables and cell measures “supplementary variables”. Some *preprocessor functions* need this information to work. For example, the *area_statistics* preprocessor function needs to know area of each grid cell in order to compute a correctly weighted statistic.

To attach these variables to a dataset, the `supplementary_variables` keyword can be used. For example, to add cell area to a dataset, it can be specified as follows:

```
datasets:
- dataset: BCC-ESM1
  project: CMIP6
  exp: historical
  ensemble: r1i1p1f1
  grid: gn
```

(continues on next page)

(continued from previous page)

```

supplementary_variables:
  - short_name: areacella
    mip: fx
    exp: 1pctCO2

```

Note that the supplementary variable will inherit the facet values from the main dataset, so only those facet values that differ need to be specified.

Automatically selecting the supplementary dataset

When using many datasets, it may be quite a bit of work to find out which facet values are required to find the corresponding supplementary data. The tool can automatically guess the best matching supplementary dataset. To use this feature, the supplementary dataset can be specified as:

```

datasets:
  - dataset: BCC-ESM1
    project: CMIP6
    exp: historical
    ensemble: r1i1p1f1
    grid: gn
    supplementary_variables:
      - short_name: areacella
        mip: fx
        exp: '*'
        activity: '*'
        ensemble: '*'

```

With this syntax, the tool will search all available values of `exp`, `activity`, and `ensemble` and use the supplementary dataset that shares the most facet values with the main dataset. Note that this behaviour is different from *using wildcards in the main dataset*, where they will be expanded to generate all matching datasets. The available datasets are shown in the debug log messages when running a recipe with wildcards, so if a different supplementary dataset is preferred, these messages can be used to see what facet values are available. The facet values for local files are retrieved from the directory tree where the directories represent the facets values. Reading facet values from file names is not yet supported. If wildcard expansion fails, this is because there were either no files found that match those facets, or the facets could not be read from the directory name or ESGF.

Automatic definition of supplementary variables

If an ancillary variable or cell measure is *needed by a preprocessor function*, but it is not specified in the recipe, the tool will automatically make a best guess using the syntax above. Usually this will work fine, but if it does not, it is recommended to explicitly define the supplementary variables in the recipe.

To disable this automatic addition, define the supplementary variable as usual, but add the special facet `skip` with value `true`. See *Supplementary variables (ancillary variables and cell measures)* for an example recipe.

Saving ancillary variables and cell measures

By default, ancillary variables and cell measures will be removed from the main variable before saving it to file because they can be as big as the main variable. To keep the supplementary variables, disable the preprocessor function that removes them by setting `remove_supplementary_variables: false` in the preprocessor profile in the recipe.

Concatenating data corresponding to multiple facets

It is possible to define the experiment as a list to concatenate two experiments. Here it is an example concatenating the *historical* experiment with *rcp85*

```
datasets:
- {dataset: CanESM2, project: CMIP5, exp: [historical, rcp85], ensemble: r1i1p1, start_
↪year: 2001, end_year: 2004}
```

It is also possible to define the ensemble as a list when the two experiments have different ensemble names. In this case, the specified datasets are concatenated into a single cube:

```
datasets:
- {dataset: CanESM2, project: CMIP5, exp: [historical, rcp85], ensemble: [r1i1p1,
↪r1i2p1], start_year: 2001, end_year: 2004}
```

Short notation of ensemble members and sub-experiments

ESMValTool also supports a simplified syntax to add multiple ensemble members from the same dataset. In the ensemble key, any element in the form $(x:y)$ will be replaced with all numbers from x to y (both inclusive), adding a dataset entry for each replacement. For example, to add ensemble members *r1i1p1* to *r10i1p1* you can use the following abbreviated syntax:

```
datasets:
- {dataset: CanESM2, project: CMIP5, exp: historical, ensemble: "r(1:10)i1p1", start_
↪year: 2001, end_year: 2004}
```

It can be included multiple times in one definition. For example, to generate the datasets definitions for the ensemble members *r1i1p1* to *r5i1p1* and from *r1i2p1* to *r5i1p1* you can use:

```
datasets:
- {dataset: CanESM2, project: CMIP5, exp: historical, ensemble: "r(1:5)i(1:2)p1",
↪start_year: 2001, end_year: 2004}
```

Please, bear in mind that this syntax can only be used in the ensemble tag. Also, note that the combination of multiple experiments and ensembles, like `exp: [historical, rcp85], ensemble: [r1i1p1, "r(2:3)i1p1"]` is not supported and will raise an error.

The same simplified syntax can be used to add multiple sub-experiments:

```
datasets:
- {dataset: MIROC6, project: CMIP6, exp: dcppA-hindcast, ensemble: r1i1p1f1, sub_
↪experiment: s(2000:2002), grid: gn, start_year: 2003, end_year: 2004}
```

Time ranges

When using the `timerange` tag to specify the start and end points, possible values can be as follows:

timerange	effect
'1980/1982'	Spans from 01/01/1980 to 31/12/1982
'198002/198205'	Spans from 01/02/1980 to 31/05/1982
'19800302/19820403'	Spans from 02/03/1980 to 03/04/1982
'19800504T100000/19800504T110000'	Spans from 04/05/1980 at 10h to 11h
'1980/P5Y'	Starting from 01/01/1980, spans 5 years
'P2Y5M/198202'	Ending at 28/02/1982, spans 2 years and 5 months
'*'	Finds all available years
'*/1982'	Finds first available point, spans to 31/12/1982
'*/P6Y'	Finds first available point, spans 6 years from it
'198003/*'	Starting from 01/03/1980, spans until the last available point
'P5M/*'	Finds last available point, spans 5 months backwards from it

Note

Please make sure to use a consistent number of digits for the start and end point when using `timerange`, e.g., instead of `198005/2000`, use `198005/200012`. Otherwise, it might happen that ESMValTool does not find your data even though the corresponding years are available. This also applies to wildcards: Wildcards are usually resolved using the `timerange` in the file name. If this is given in the form `YYYYMM`, then the other time point in `timerange` needs to be in the same format, e.g., use `*/200012` instead of `*/2000` in this case. If you use wildcards and get an unexpected error about missing data, have a look at the resolved `timerange` in the error message (`ERROR No input files found for variable {'timerange': '197901/2000', ...}`) and make sure that the number of digits in it is consistent.

Note that this section is not required, as datasets can also be provided in the *Diagnostics* section.

3.1.3 Recipe section: preprocessors

The preprocessor section of the recipe includes one or more preprocessors, each of which may call the execution of one or several preprocessor functions.

Each preprocessor section includes:

- A preprocessor name (any name, under `preprocessors`);
- A list of preprocessor steps to be executed (choose from the API);
- Any or none arguments given to the preprocessor steps;
- The order that the preprocessor steps are applied can also be specified using the `custom_order` preprocessor function.

The following snippet is an example of a preprocessor named `prep_map` that contains multiple preprocessing steps (*Horizontal regridding* with two arguments, *Time manipulation* with no arguments (i.e., calculating the average over the time dimension) and *Multi-model statistics* with two arguments):

```
preprocessors:
  prep_map:
    regrid:
      target_grid: 1x1
```

(continues on next page)

(continued from previous page)

```

scheme: linear
climate_statistics:
  operator: mean
multi_model_statistics:
  span: overlap
  statistics: [mean]

```

Note

In this case no `preprocessors` section is needed the workflow will apply a default preprocessor consisting of only basic operations like: loading data, applying CMOR checks and fixes (*CMORization and dataset-specific fixes*) and saving the data to disk.

Preprocessor operations will be applied using the default order as listed in *Preprocessor functions*. Preprocessor tasks can be set to run in the order they are listed in the recipe by adding `custom_order: true` to the preprocessor definition.

3.1.4 Recipe section: diagnostics

The diagnostics section includes one or more diagnostics. Each diagnostic section will include:

- the *variable(s)* to preprocess, including the preprocessor to be applied to each variable;
- the diagnostic script(s) to be run;
- a description of the diagnostic and lists of themes and realms that it applies to;
- an optional `additional_datasets` section.
- an optional title and description, used to generate the title and description in the `index.html` output file.

The diagnostics section defines tasks

The diagnostic section(s) define the tasks that will be executed when running the recipe. For each variable a preprocessing task will be defined and for each diagnostic script a diagnostic task will be defined. These tasks can be viewed in the `main_log_debug.txt` file that is produced every run. Each task has a unique name that defines the subdirectory where the results of that task are stored. Task names start with the name of the diagnostic section followed by a `'` and then the name of the variable section for a preprocessing task or the name of the diagnostic script section for a diagnostic task.

A (simplified) example diagnostics section could look like

```

diagnostics:
  diagnostic_name:
    title: Air temperature tutorial diagnostic
    description: A longer description can be added here.
    themes:
      - phys
    realms:
      - atmos
    variables:
      variable_name:
        short_name: ta
        preprocessor: preprocessor_name

```

(continues on next page)

(continued from previous page)

```

    mip: Amon
  scripts:
    script_name:
      script: examples/ diagnostic.py

```

Note that the example recipe above contains a single diagnostic section called `diagnostic_name` and will result in two tasks:

- a preprocessing task called `diagnostic_name/variable_name` that will preprocess air temperature data for each dataset in the *Datasets* section of the recipe (not shown).
- a diagnostic task called `diagnostic_name/script_name`

The path to the script provided in the `script` option should be either the absolute path to the script, or the path relative to the `esmvaltool/diag_scripts` directory.

Depending on the installation configuration, you may get an error of “file does not exist” when the system tries to run the diagnostic script using relative paths. If this happens, use an absolute path instead.

Note that the script should either have the extension for a supported language, i.e. `.py`, `.R`, or `.ncl` for Python, R, and NCL diagnostics respectively, or be executable if it is written in any other language.

Ancestor tasks

Some tasks require the result of other tasks to be ready before they can start, e.g. a diagnostic script needs the preprocessed variable data to start. Thus each task has zero or more ancestor tasks. By default, each diagnostic task in a diagnostic section has all variable preprocessing tasks in that same section as ancestors. However, this can be changed using the `ancestors` keyword. Note that wildcard expansion can be used to define ancestors.

```

diagnostics:
  diagnostic_1:
    variables:
      airtemp:
        short_name: ta
        preprocessor: preprocessor_name
        mip: Amon
    scripts:
      script_a:
        script: diagnostic_a.py
  diagnostic_2:
    variables:
      precip:
        short_name: pr
        preprocessor: preprocessor_name
        mip: Amon
    scripts:
      script_b:
        script: diagnostic_b.py
        ancestors: [diagnostic_1/script_a, precip]

```

The example recipe above will result in four tasks:

- a preprocessing task called `diagnostic_1/airtemp`
- a diagnostic task called `diagnostic_1/script_a`
- a preprocessing task called `diagnostic_2/precip`

- a diagnostic task called `diagnostic_2/script_b`

the preprocessing tasks do not have any ancestors, while the `diagnostic_a.py` script will receive the preprocessed air temperature data (has ancestor `diagnostic_1/airtemp`) and the `diagnostic_b.py` script will receive the results of `diagnostic_a.py` and the preprocessed precipitation data (has ancestors `diagnostic_1/script_a` and `diagnostic_2/precip`).

Task priority

Tasks are assigned a priority, with tasks appearing earlier on in the recipe getting higher priority. The tasks will be executed sequentially or in parallel, depending on the *configuration option* `max_parallel_tasks`. When there are fewer than `max_parallel_tasks` running, tasks will be started according to their priority. For obvious reasons, only tasks that are not waiting for ancestor tasks can be started. This feature makes it possible to reduce the processing time of recipes with many tasks, by placing tasks that take relatively long near the top of the recipe. Of course this only works when settings `max_parallel_tasks` to a value larger than 1. The current priority and run time of individual tasks can be seen in the log messages shown when running the tool (a lower number means higher priority).

Variable and dataset definitions

To define a *variable/dataset* combination that corresponds to an actual variable from a dataset, the keys in each variable section are combined with the keys of each dataset definition. If two versions of the same key are provided, then the key in the datasets section will take precedence over the keys in variables section. For many recipes it makes more sense to define the `start_year` and `end_year` items in the variable section, because the diagnostic script assumes that all the data has the same time range.

Variable short names usually do not change between datasets supported by ESMValCore, as they are usually changed to match CMIP. Nevertheless, there are small changes in variable names in CMIP6 with respect to CMIP5 (i.e. sea ice concentration changed from `sic` to `siconc`). ESMValCore is aware of some of them and can do the automatic translation when needed. It will even do the translation in the preprocessed file so the diagnostic does not have to deal with this complexity, setting the short name in all files to match the one used by the recipe. For example, if `sic` is requested, ESMValCore will find `sic` or `siconc` depending on the project, but all preprocessed files while use `sic` as their `short_name`. If the recipe requested `siconc`, the preprocessed files will be identical except that they will use the `short_name siconc` instead.

Diagnostic and variable specific datasets

The `additional_datasets` option can be used to add datasets beyond those listed in the *Datasets* section. This is useful if specific datasets need to be used only by a specific diagnostic or variable, i.e. it can be added both at diagnostic level, where it will apply to all variables in that diagnostic section or at individual variable level. For example, this can be a good way to add observational datasets, which are usually variable-specific.

Running a simple diagnostic

The following example, taken from `recipe_ocean_example.yml`, shows a diagnostic named `diag_map`, which loads the temperature at the ocean surface between the years 2001 and 2003 and then passes it to the `prep_map` preprocessor. The result of this process is then passed to the ocean diagnostic map script, `ocean/diagnostic_maps.py`.

```
diagnostics:
  diag_map:
    title: Global Ocean Surface regridded temperature map
    description: Add a longer description here.
    variables:
      tos: # Temperature at the ocean surface
        preprocessor: prep_map
        start_year: 2001
```

(continues on next page)

(continued from previous page)

```

    end_year: 2003
  scripts:
    Global_Ocean_Surface_regrid_map:
      script: ocean/diagnostic_maps.py

```

Passing arguments to a diagnostic script

The diagnostic script section(s) may include custom arguments that can be used by the diagnostic script; these arguments are stored at runtime in a dictionary that is then made available to the diagnostic script via the interface link, independent of the language the diagnostic script is written in. Here is an example of such groups of arguments:

```

scripts:
  autoassess_strato_test_1: &autoassess_strato_test_1_settings
    script: autoassess/autoassess_area_base.py
    title: "Autoassess Stratosphere Diagnostic Metric MPI-MPI"
    area: stratosphere
    control_model: MPI-ESM-LR
    exp_model: MPI-ESM-MR
    obs_models: [ERA-Interim] # list to hold models that are NOT for metrics but for_
↳obs operations
    additional_metrics: [ERA-Interim, inmcm4] # list to hold additional datasets for_
↳metrics

```

In this example, apart from specifying the diagnostic script `script: autoassess/autoassess_area_base.py`, we pass a suite of parameters to be used by the script (`area`, `control_model` etc). These parameters are stored in key-value pairs in the diagnostic configuration file, an interface file that can be used by importing the `run_diagnostic` utility:

```

from esmvaltool.diag_scripts.shared import run_diagnostic

# write the diagnostic code here e.g.
def run_some_diagnostic(my_area, my_control_model, my_exp_model):
    """Diagnostic to be run."""
    if my_area == 'stratosphere':
        diag = my_control_model / my_exp_model
        return diag

def main(cfg):
    """Main diagnostic run function."""
    my_area = cfg['area']
    my_control_model = cfg['control_model']
    my_exp_model = cfg['exp_model']
    run_some_diagnostic(my_area, my_control_model, my_exp_model)

if __name__ == '__main__':

    with run_diagnostic() as config:
        main(config)

```

This way a lot of the optional arguments necessary to a diagnostic are at the user's control via the recipe.

Running your own diagnostic

If the user wants to test a newly-developed `my_first_diagnostic.py` which is not yet part of the ESMValTool diagnostics library, he/she do it by passing the absolute path to the diagnostic:

```
diagnostics:
  myFirstDiag:
    title: Let's do some science!
    description: John Doe wrote a funny diagnostic
    variables:
      tos: # Temperature at the ocean surface
      preprocessor: prep_map
      start_year: 2001
      end_year: 2003
    scripts:
      JoeDiagFunny:
        script: /home/users/john_doe/esmvaltool_testing/my_first_diagnostic.py
```

This way the user may test a new diagnostic thoroughly before committing to the GitHub repository and including it in the ESMValTool diagnostics library.

Reusing parameters from one script to another

Due to yaml features it is possible to recycle entire diagnostics sections for use with other diagnostics. Here is an example:

```
scripts:
  cycle: &cycle_settings
    script: perfmetrics/main.ncl
    plot_type: cycle
    time_avg: monthlyclim
  grading: &grading_settings
  <<: *cycle_settings
    plot_type: cycle_latlon
    calc_grading: true
    normalization: [centered_median, none]
```

In this example the hook `&cycle_settings` can be used to pass the `cycle:` parameters to `grading:` via the shortcut `<<: *cycle_settings`.

3.2 Preprocessor

In this section, each of the preprocessor modules is described, roughly following the default order in which preprocessor functions are applied:

- *Overview*
- *Statistical preprocessors*
- *Variable derivation*
- *CMORization and dataset-specific fixes*
- *Supplementary variables (ancillary variables and cell measures)*
- *Vertical interpolation*

- *Weighting*
- *Land-sea masking*
- *Horizontal regridding*
- *Missing values masks*
- *Ensemble statistics*
- *Multi-model statistics*
- *Time manipulation*
- *Area manipulation*
- *Volume manipulation*
- *Cycles*
- *Trend*
- *Detrend*
- *Rolling window statistics*
- *Unit conversion*
- *Comparison with reference dataset*
- *Other*

See *Preprocessor functions* for implementation details and the exact default order.

3.2.1 Overview

The ESMValCore preprocessor can be used to perform a broad range of operations on the input data before diagnostics or metrics are applied. The preprocessor performs these operations in a centralized, documented and efficient way, thus reducing the data processing load on the diagnostics side. For an overview of the preprocessor structure see the *Recipe section: preprocessors*.

Each of the preprocessor operations is written in a dedicated python module and all of them receive and return an instance of `iris.cube.Cube`, working sequentially on the data with no interactions between them. The order in which the preprocessor operations is applied is set by default to minimize the loss of information due to, for example, temporal and spatial subsetting or multi-model averaging. Nevertheless, the user is free to change such order to address specific scientific requirements, but keeping in mind that some operations must be necessarily performed in a specific order. This is the case, for instance, for multi-model statistics, which required the model to be on a common grid and therefore has to be called after the regridding module.

3.2.2 Statistical preprocessors

Many preprocessors calculate statistics over data. Those preprocessors typically end with `_statistics`, e.g., `area_statistics()` or `multi_model_statistics()`. All these preprocessors support the options *operator*, which directly correspond to `iris.analysis.Aggregator` objects used to perform the statistical calculations. In addition, arbitrary keyword arguments can be passed which are directly passed to the corresponding `iris.analysis.Aggregator` object.

Note

The preprocessors `multi_model_statistics()` and `ensemble_statistics()` support the computation of multiple statistics at the same time. In these cases, they are defined by the option *statistics* (instead of *operator*), which takes a list of possible operators. Each operator can be given as single string or as dictionary. In the

latter case, the dictionary needs the keyword *operator* (corresponding to the *operator* as above). All other keywords are interpreted as keyword arguments for the given operator.

Some operators support weights for some preprocessors (see following table), which are used by default. The following operators are currently fully supported; other operators might be supported too if proper keyword arguments are specified:

<i>operator</i>	Corresponding Aggregator	Weighted? ¹
<code>gmean</code>	<code>iris.analysis.GMEAN</code>	no
<code>hmean</code>	<code>iris.analysis.HMEAN</code>	no
<code>max</code>	<code>iris.analysis.MAX</code>	no
<code>mean</code>	<code>iris.analysis.MEAN</code>	yes
<code>median</code>	<code>iris.analysis.MEDIAN</code> ²	no
<code>min</code>	<code>iris.analysis.MIN</code>	no
<code>peak</code>	<code>iris.analysis.PEAK</code>	no
<code>percentile</code>	<code>iris.analysis.PERCENTILE</code>	no
<code>rms</code>	<code>iris.analysis.RMS</code>	yes
<code>std_dev</code>	<code>iris.analysis.STD_DEV</code>	no
<code>sum</code>	<code>iris.analysis.SUM</code>	yes
<code>variance</code>	<code>iris.analysis.VARIANCE</code>	no
<code>wpercentile</code>	<code>iris.analysis.WPERCENTILE</code>	yes

Examples

Calculate the global (weighted) mean:

```
preprocessors:
  global_mean:
    area_statistics:
      operator: mean
```

Calculate zonal maximum.

```
preprocessors:
  zonal_max:
    zonal_statistics:
      operator: max
```

Calculate the 95% percentile over each month separately (will result in 12 time steps, one for January, one for February, etc.):

```
preprocessors:
  monthly_percentiles:
    climate_statistics:
      period: monthly
```

(continues on next page)

¹ The following preprocessor support weighted statistics by default: `area_statistics()`: weighted by grid cell areas (see also *Supplementary variables (ancillary variables and cell measures)*); `climate_statistics()`: weighted by lengths of time intervals; `volume_statistics()`: weighted by grid cell volumes (see also *Supplementary variables (ancillary variables and cell measures)*); `axis_statistics()`: weighted by corresponding coordinate bounds.

² `iris.analysis.MEDIAN` is not lazy, but much faster than `iris.analysis.PERCENTILE`. For a lazy median, use `percentile` with the keyword argument `percent = 50`.

(continued from previous page)

```
operator: percentile
percent: 95.0
```

Calculate multi-model median, 5%, and 95% percentiles:

```
preprocessors:
  mm_stats:
    multi_model_statistics:
      span: overlap
      statistics:
        - operator: percentile
          percent: 5
        - operator: median
        - operator: percentile
          percent: 95
```

Calculate the global non-weighted root mean square:

```
preprocessors:
  global_mean:
    area_statistics:
      operator: rms
      weights: false
```

Warning

The disabling of weights by specifying the keyword argument `weights: false` needs to be used with great care; from a scientific standpoint, we strongly recommend to **not** use it!

3.2.3 Variable derivation

The variable derivation module allows to derive variables which are not in the CMIP standard data request using standard variables as input. The typical use case of this operation is the evaluation of a variable which is only available in an observational dataset but not in the models. In this case a derivation function is provided by the ESMValCore in order to calculate the variable and perform the comparison. For example, several observational datasets deliver total column ozone as observed variable (`toz`), but CMIP models only provide the ozone 3D field. In this case, a derivation function is provided to vertically integrate the ozone and obtain total column ozone for direct comparison with the observations.

By default, the variable derivation will be applied only if the variable is not already available in the input data, but the derivation can be forced by setting the `force_derivation` flag.

```
variables:
  toz:
    derive: true
    force_derivation: false
```

The required arguments for this module are two boolean switches:

- `derive`: activate variable derivation
- `force_derivation`: force variable derivation even if the variable is directly available in the input data.

See also `esmvalcore.preprocessor.derive()`. To get an overview on derivation scripts and how to implement new ones, please go to *Deriving a variable*.

3.2.4 CMORization and dataset-specific fixes

Data checking

Data preprocessed by ESMValCore is automatically checked against its CMOR definition. To reduce the impact of this check while maintaining it as reliable as possible, it is split in two parts: one will check the metadata and will be done just after loading and concatenating the data and the other one will check the data itself and will be applied after all extracting operations are applied to reduce the amount of data to process.

Checks include, but are not limited to:

- Requested coordinates are present and comply with their definition.
- Correctness of variable names, units and other metadata.
- Compliance with the valid minimum and maximum values allowed if defined.

The most relevant (i.e. a missing coordinate) will raise an error while others (i.e an incorrect long name) will be reported as a warning.

Some of those issues will be fixed automatically by the tool, including the following:

- Incorrect standard or long names.
- Incorrect units, if they can be converted to the correct ones.
- Direction of coordinates.
- Automatic clipping of longitude to 0 - 360 interval.
- Minute differences between the required and actual vertical coordinate values

Dataset specific fixes

Sometimes, the checker will detect errors that it can not fix by itself. ESMValCore deals with those issues by applying specific fixes for those datasets that require them. Fixes are applied at three different preprocessor steps:

- `fix_file`: apply fixes to data before loading them with Iris. This is mainly intended to fix errors that prevent data loading with Iris (e.g., those related to `missing_value` or `_FillValue`) or operations that are more efficient with other packages (e.g., loading files with lots of variables is much faster with Xarray than Iris). See `esmvalcore.preprocessor.fix_file()`.
- `fix_metadata`: metadata fixes are done just before concatenating the cubes loaded from different files in the final one. Automatic metadata fixes are also applied at this step. See `esmvalcore.preprocessor.fix_metadata()`.
- `fix_data`: data fixes are applied before starting any operation that will alter the data itself. Automatic data fixes are also applied at this step. See `esmvalcore.preprocessor.fix_data()`.

To get an overview on data fixes and how to implement new ones, please go to *Fixing data*.

3.2.5 Supplementary variables (ancillary variables and cell measures)

The following preprocessor functions either require or prefer using an *ancillary variable* or *cell measure* to perform their computations. In ESMValCore we call both types of variables “supplementary variables”.

Preprocessor	Variable short name	Variable standard name
<i>area_statistics</i> ⁴	areacella, areacello	cell_area
<i>mask_landsea</i> ⁴	sftlf, sftof	land_area_fraction, sea_area_fraction
<i>mask_landseaice</i> ³	sftgif	land_ice_area_fraction
<i>volume_statistics</i> ⁴	volcello, areacello	ocean_volume, cell_area
<i>weighting_landsea_fraction</i> ³	sftlf, sftof	land_area_fraction, sea_area_fraction
<i>distance_metric</i> ⁵	areacella, areacello	cell_area
<i>histogram</i> ⁵	areacella, areacello	cell_area
<i>extract_surface_from_atm</i> ³	ps	surface_air_pressure

Only one of the listed variables is required. Supplementary variables can be defined in the recipe as described in *Defining supplementary variables (ancillary variables and cell measures)*. If the automatic selection does not give the desired result, specify the supplementary variables in the recipe as described in *Defining supplementary variables (ancillary variables and cell measures)*.

By default, supplementary variables will be removed from the variable before saving it to file because they can be as big as the main variable. To keep the supplementary variables, disable the preprocessor function `esmvalcore.preprocessor.remove_supplementary_variables()` that removes them by setting `remove_supplementary_variables: false` in the preprocessor in the recipe.

Examples

Compute the global mean surface air temperature, while *automatically selecting the best matching supplementary dataset*:

```

datasets:
- dataset: BCC-ESM1
  project: CMIP6
  ensemble: r1i1p1f1
  grid: gn
- dataset: MPI-ESM-MR
  project: CMIP5
  ensemble: r1i1p1,

preprocessors:
  global_mean:
    area_statistics:
      operator: mean

diagnostics:
  example_diagnostic:
    description: Global mean temperature.
    variables:
      tas:
        mip: Amon
        preprocessor: global_mean
        exp: historical
        timerange: '1990/2000'

```

(continues on next page)

⁴ This preprocessor prefers at least one of the mentioned supplementary variables. If none is defined in the recipe, automatically look for them. If none is found, a warning will be raised (but no error).

³ This preprocessor requires at least one of the mentioned supplementary variables. If none is defined in the recipe, automatically look for them. If none is found, an error will be raised.

⁵ This preprocessor optionally takes one of the mentioned supplementary variables. If none is defined in the recipe, none is added.

(continued from previous page)

```

supplementary_variables:
  - short_name: areacella
    mip: fx
    exp: '*'
    activity: '*'
    ensemble: '*'
scripts: null

```

Attach the land area fraction as an ancillary variable to surface air temperature and store both in the same file:

```

datasets:
  - dataset: BCC-ESM1
    ensemble: r1i1p1f1
    grid: gn

preprocessors:
  keep_land_area_fraction:
    remove_supplementary_variables: false

diagnostics:
  example_diagnostic:
    description: Attach land area fraction.
    variables:
      tas:
        mip: Amon
        project: CMIP6
        preprocessor: keep_land_area_fraction
        exp: historical
        timerange: '1990/2000'
        supplementary_variables:
          - short_name: sftlf
            mip: fx
            exp: 1pctCO2
    scripts: null

```

Automatically define the required ancillary variable (sftlf in this case) and cell measure (areacella), but do not use areacella for dataset BCC-ESM1:

```

datasets:
  - dataset: BCC-ESM1
    project: CMIP6
    ensemble: r1i1p1f1
    grid: gn
    supplementary_variables:
      - short_name: areacella
        skip: true
  - dataset: MPI-ESM-MR
    project: CMIP5
    ensemble: r1i1p1

preprocessors:
  global_land_mean:

```

(continues on next page)

(continued from previous page)

```

mask_landsea:
  mask_out: sea
area_statistics:
  operator: mean

diagnostics:
  example_diagnostic:
    description: Global mean temperature.
    variables:
      tas:
        mip: Amon
        preprocessor: global_land_mean
        exp: historical
        timerange: '1990/2000'
    scripts: null

```

3.2.6 Vertical interpolation

Vertical level selection is an important aspect of data preprocessing since it allows the scientist to perform a number of metrics specific to certain levels (whether it be air pressure or depth, e.g. the Quasi-Biennial-Oscillation (QBO) u30 is computed at 30 hPa). Dataset native vertical grids may not come with the desired set of levels, so an interpolation operation will be needed to regrid the data vertically. ESMValCore can perform this vertical interpolation via the `extract_levels` preprocessor. Level extraction may be done in a number of ways.

Level extraction can be done at specific values passed to `extract_levels` as `levels:` with its value a list of levels (note that the units are CMOR-standard, Pascals (Pa)):

```

preprocessors:
  preproc_select_levels_from_list:
    extract_levels:
      levels: [100000., 50000., 3000., 1000.]
      scheme: linear

```

It is also possible to extract the CMIP-specific, CMOR levels as they appear in the CMOR table, e.g. `plev10` or `plev17` or `plev19` etc:

```

preprocessors:
  preproc_select_levels_from_cmip_table:
    extract_levels:
      levels: {cmor_table: CMIP6, coordinate: plev10}
      scheme: nearest

```

Of good use is also the level extraction with values specific to a certain dataset, without the user actually polling the dataset of interest to find out the specific levels: e.g. in the example below we offer two alternatives to extract the levels and vertically regrid onto the vertical levels of ERA-Interim:

```

preprocessors:
  preproc_select_levels_from_dataset:
    extract_levels:
      levels: ERA-Interim
      # This also works, but allows specifying the pressure coordinate name
      # levels: {dataset: ERA-Interim, coordinate: air_pressure}
      scheme: linear_extrapolate

```

By default, vertical interpolation is performed in the dimension coordinate of the z axis. If you want to explicitly declare the z axis coordinate to use (for example, `air_pressure` in variables that are provided in model levels and not pressure levels) you can override that automatic choice by providing the name of the desired coordinate:

```
preprocessors:
  preproc_select_levels_from_dataset:
    extract_levels:
      levels: ERA-Interim
      scheme: linear_extrapolate
      coordinate: air_pressure
```

If `coordinate` is specified, pressure levels (if present) can be converted to height levels and vice versa using the US standard atmosphere. E.g. `coordinate = altitude` will convert existing pressure levels (`air_pressure`) to height levels (`altitude`); `coordinate = air_pressure` will convert existing height levels (`altitude`) to pressure levels (`air_pressure`).

If the requested levels are very close to the values in the input data, the function will just select the available levels instead of interpolating. The meaning of ‘very close’ can be changed by providing the parameters:

- **rtol**
Relative tolerance for comparing the levels in the input data to the requested levels. If the levels are sufficiently close, the requested levels will be assigned to the vertical coordinate and no interpolation will take place. The default value is 10^{-7} .
- **atol**
Absolute tolerance for comparing the levels in the input data to the requested levels. If the levels are sufficiently close, the requested levels will be assigned to the vertical coordinate and no interpolation will take place. By default, `atol` will be set to 10^{-7} times the mean value of of the available levels.

Schemes for vertical interpolation and extrapolation

The vertical interpolation currently supports the following schemes:

- **linear**: Linear interpolation without extrapolation, i.e., extrapolation points will be masked even if the source data is not a masked array.
- **linear_extrapolate**: Linear interpolation with **nearest-neighbour** extrapolation, i.e., extrapolation points will take their value from the nearest source point.
- **nearest**: Nearest-neighbour interpolation without extrapolation, i.e., extrapolation points will be masked even if the source data is not a masked array.
- **nearest_extrapolate**: Nearest-neighbour interpolation with nearest-neighbour extrapolation, i.e., extrapolation points will take their value from the nearest source point.
- See also `esmvalcore.preprocessor.extract_levels()`.
- See also `esmvalcore.preprocessor.get_cmor_levels()`.

Note

Controlling the extrapolation mode allows us to avoid situations where extrapolating values makes little physical sense (e.g. extrapolating beyond the last data point).

3.2.7 Weighting

Land/sea fraction weighting

This preprocessor allows weighting of data by land or sea fractions. In other words, this function multiplies the given input field by a fraction in the range 0-1 to account for the fact that not all grid points are completely land- or sea-covered.

The application of this preprocessor is very important for most carbon cycle variables (and other land surface outputs), which are e.g. reported in units of $kgC\ m^{-2}$. Here, the surface unit actually refers to 'square meter of land/sea' and NOT 'square meter of gridbox'. In order to integrate these globally or regionally one has to weight by both the surface quantity and the land/sea fraction.

For example, to weight an input field with the land fraction, the following preprocessor can be used:

```
preprocessors:
  preproc_weighting:
    weighting_landsea_fraction:
      area_type: land
      exclude: ['CanESM2', 'reference_dataset']
```

Allowed arguments for the keyword `area_type` are `land` (fraction is 1 for grid cells with only land surface, 0 for grid cells with only sea surface and values in between 0 and 1 for coastal regions) and `sea` (1 for sea, 0 for land, in between for coastal regions). The optional argument `exclude` allows to exclude specific datasets from this preprocessor, which is for example useful for climate models which do not offer land/sea fraction files. This arguments also accepts the special dataset specifiers `reference_dataset` and `alternative_dataset`.

This function requires a land or sea area fraction *ancillary variable*. This supplementary variable, either `sftlf` or `sftof`, should be attached to the main dataset as described in *Defining supplementary variables (ancillary variables and cell measures)*.

See also `esmvalcore.preprocessor.weighting_landsea_fraction()`.

3.2.8 Masking

Introduction to masking

Certain metrics and diagnostics need to be computed and performed on specific domains on the globe. The preprocessor supports filtering the input data on continents, oceans/seas and ice. This is achieved by masking the model data and keeping only the values associated with grid points that correspond to, e.g., land, ocean or ice surfaces, as specified by the user. Where possible, the masking is realized using the standard mask files provided together with the model data as part of the CMIP data request (the so-called ancillary variable). In the absence of these files, the Natural Earth masks are used: although these are not model-specific, they represent a good approximation since they have a much higher resolution than most of the models and they are regularly updated with changing geographical features.

Land-sea masking

To mask out a certain domain (e.g., sea) in the preprocessor, `mask_landsea` can be used:

```
preprocessors:
  preproc_mask:
    mask_landsea:
      mask_out: sea
```

and requires only one argument: `mask_out`: either `land` or `sea`.

This function prefers using a land or sea area fraction *ancillary variable*, but if it is not available it will compute a mask based on *Natural Earth* shapefiles. This supplementary variable, either `sftlf` or `sftof`, can be attached to the main dataset as described in *Defining supplementary variables (ancillary variables and cell measures)*.

If the corresponding ancillary variable is not available (which is the case for some models and almost all observational datasets), the preprocessor attempts to mask the data using Natural Earth mask files (that are vectorized rasters). As mentioned above, the spatial resolution of the the Natural Earth masks are much higher than any typical global model (10m for land and glaciated areas and 50m for ocean masks).

See also `esmvalcore.preprocessor.mask_landsea()`.

Ice masking

For masking out ice sheets, the preprocessor uses a different function, to ensure that both land and sea or ice can be masked out without losing generality. To mask ice out, `mask_landseaice` can be used:

```
preprocessors:
  preproc_mask:
    mask_landseaice:
      mask_out: ice
```

and requires only one argument: `mask_out`: either `landsea` or `ice`.

This function requires a land ice area fraction *ancillary variable*. This supplementary variable `sftgif` should be attached to the main dataset as described in *Defining supplementary variables (ancillary variables and cell measures)*.

See also `esmvalcore.preprocessor.mask_landseaice()`.

Glaciated masking

For masking out glaciated areas a Natural Earth shapefile is used. To mask glaciated areas out, `mask_glaciated` can be used:

```
preprocessors:
  preproc_mask:
    mask_glaciated:
      mask_out: glaciated
```

and it requires only one argument: `mask_out`: only `glaciated`.

See also `esmvalcore.preprocessor.mask_landseaice()`.

Missing values masks

Missing (masked) values can be a nuisance especially when dealing with multi-model ensembles and having to compute multi-model statistics; different numbers of missing data from dataset to dataset may introduce biases and artificially assign more weight to the datasets that have less missing data. This is handled via the missing values masks: two types of such masks are available, one for the multi-model case and another for the single model case.

The multi-model missing values mask (`mask_fillvalues`) is a preprocessor step that usually comes after all the single-model steps (regridding, area selection etc) have been performed; in a nutshell, it combines missing values masks from individual models into a multi-model missing values mask; the individual model masks are built according to common criteria: the user chooses a time window in which missing data points are counted, and if the number of missing data points relative to the number of total data points in a window is less than a chosen fractional threshold, the window is discarded i.e. all the points in the window are masked (set to missing).

```
preprocessors:
  missing_values_preprocessor:
    mask_fillvalues:
      threshold_fraction: 0.95
      min_value: 19.0
      time_window: 10.0
```

In the example above, the fractional threshold for missing data vs. total data is set to 95% and the time window is set to 10.0 (units of the time coordinate units). Optionally, a minimum value threshold can be applied, in this case it is set to 19.0 (in units of the variable units).

See also `esmvalcore.preprocessor.mask_fillvalues()`.

Common mask for multiple models

To create a combined multi-model mask (all the masks from all the analyzed datasets combined into a single mask using a logical OR), the preprocessor `mask_multimodel` can be used. In contrast to `mask_fillvalues`, `mask_multimodel` does not expect that the datasets have a time coordinate, but works on datasets with arbitrary (but identical) coordinates. After `mask_multimodel`, all involved datasets have an identical mask.

See also `esmvalcore.preprocessor.mask_multimodel()`.

Minimum, maximum and interval masking

Thresholding on minimum and maximum accepted data values can also be performed: masks are constructed based on the results of thresholding; inside and outside interval thresholding and masking can also be performed. These functions are `mask_above_threshold`, `mask_below_threshold`, `mask_inside_range`, and `mask_outside_range`.

These functions always take a cube as first argument and either `threshold` for threshold masking or the pair `minimum`, `maximum` for interval masking.

See also `esmvalcore.preprocessor.mask_above_threshold()` and related functions.

3.2.9 Horizontal regridding

Regridding is necessary when various datasets are available on a variety of *lat-lon* grids and they need to be brought together on a common grid (for various statistical operations e.g. multi-model statistics or for e.g. direct inter-comparison or comparison with observational datasets). Regridding is conceptually a very similar process to interpolation (in fact, the regridding engine uses interpolation and extrapolation, with various schemes). The primary difference is that interpolation is based on sample data points, while regridding is based on the horizontal grid of another cube (the reference grid). If the horizontal grids of a cube and its reference grid are sufficiently the same, regridding is automatically and silently skipped for performance reasons.

The use of the horizontal regridding functionality is flexible depending on what type of reference grid and what interpolation scheme is preferred. Below we show a few examples.

Regridding on a reference dataset grid

The example below shows how to regrid on the reference dataset ERA-Interim (observational data, but just as well CMIP, obs4MIPs, or ana4MIPs datasets can be used); in this case the *scheme* is *linear*.

```
preprocessors:
  regrid_preprocessor:
    regrid:
      target_grid: ERA-Interim
      scheme: linear
```

Regridding on an MxN grid specification

The example below shows how to regrid on a reference grid with a cell specification of 2.5x2.5 degrees. This is similar to regridding on reference datasets, but in the previous case the reference dataset grid cell specifications are not necessarily known a priori. Regridding on an MxN cell specification is oftentimes used when operating on localized data.

```
preprocessors:
  regrid_preprocessor:
    regrid:
      target_grid: 2.5x2.5
      scheme: nearest
```

In this case the nearest-neighbor interpolation scheme is used (see below for scheme definitions).

When using a MxN type of grid it is possible to offset the grid cell centrepoints using the *lat_offset* and *lon_offset* arguments:

- *lat_offset*: offsets the grid centers of the latitude coordinate w.r.t. the pole by half a grid step;
- *lon_offset*: offsets the grid centers of the longitude coordinate w.r.t. Greenwich meridian by half a grid step.

```
preprocessors:
  regrid_preprocessor:
    regrid:
      target_grid: 2.5x2.5
      lon_offset: True
      lat_offset: True
      scheme: nearest
```

Regridding to a regional target grid specification

This example shows how to regrid to a regional target grid specification. This is useful if both a *regrid* and *extract_region* step are necessary.

```
preprocessors:
  regrid_preprocessor:
    regrid:
      target_grid:
        start_longitude: 40
        end_longitude: 60
        step_longitude: 2
        start_latitude: -10
        end_latitude: 30
        step_latitude: 2
      scheme: nearest
```

This defines a grid ranging from 40° to 60° longitude with 2° steps, and -10° to 30° latitude with 2° steps. If *end_longitude* or *end_latitude* do not fall on the grid (e.g., *end_longitude*: 61), it cuts off at the nearest previous value (e.g. 60).

The longitude coordinates will wrap around the globe if necessary, i.e. *start_longitude*: 350, *end_longitude*: 370 is valid input.

The arguments are defined below:

- *start_latitude*: Latitude value of the first grid cell center (start point). The grid includes this value.
- *end_latitude*: Latitude value of the last grid cell center (end point). The grid includes this value only if it falls on a grid point. Otherwise, it cuts off at the previous value.
- *step_latitude*: Latitude distance between the centers of two neighbouring cells.
- *start_longitude*: Longitude value of the first grid cell center (start point). The grid includes this value.

- `end_longitude`: Longitude value of the last grid cell center (end point). The grid includes this value only if it falls on a grid point. Otherwise, it cuts off at the previous value.
- `step_longitude`: Longitude distance between the centers of two neighbouring cells.

Regridding input data with multiple horizontal coordinates

When there are multiple horizontal coordinates available in the input data, the standard names of the coordinates to use need to be specified. By default, these are [`latitude`, `longitude`]. To use the coordinates from a *rotated pole* grid, one would specify:

```
preprocessors:  
  regrid_preprocessor:  
    regrid:  
      target_grid: 1x1  
      scheme: linear  
      use_src_coords: [grid_latitude, grid_longitude]
```

Regridding (interpolation, extrapolation) schemes

ESMValCore provides three default regridding schemes, which are presented in *Default regridding schemes*. Additionally, it is also possible to use third party regridding schemes designed for use with `iris.cube.Cube.regrid()`. This is explained in *Generic regridding schemes*.

Grid types

In ESMValCore, we distinguish between various grid types (note that these might differ from other definitions):

- **Regular grid**: A rectilinear grid with 1D latitude and 1D longitude coordinates which are orthogonal to each other.
- **Irregular grid**: A general curvilinear grid with 2D latitude and 2D longitude coordinates with common dimensions.
- **Unstructured grid**: A grid with 1D latitude and 1D longitude coordinates with common dimensions (i.e., a simple list of points).
- **Mesh**: A mesh as supported by Iris and described in [Mesh Support](#).

Default regridding schemes

- **linear**: Bilinear regridding. For source data on a regular grid, uses `Linear` with `extrapolation_mode='mask'`. For source and/or target data on an irregular grid or mesh, uses `IrisESMFRegrid` with `method='bilinear'`. For source data on an unstructured grid, uses `UnstructuredLinear`.
- **nearest**: Nearest-neighbor regridding. For source data on a regular grid, uses `Nearest` with `extrapolation_mode='mask'`. For source and/or target data on an irregular grid or mesh, uses `IrisESMFRegrid` with `method='nearest'`. For source data on an unstructured grid, uses `UnstructuredNearest`.
- **area_weighted**: First-order conservative (area-weighted) regridding. For source data on a regular grid, uses `AreaWeighted`. For source and/or target data on an irregular grid or mesh, uses `IrisESMFRegrid` with `method='conservative'`. Source data on an unstructured grid is not supported.

Generic regridding schemes

Iris' `regridding` is based around the flexible use of so-called regridding schemes. These are classes that know how to transform a source cube with a given grid into the grid defined by a given target cube. Iris itself provides a number of useful schemes, but they are largely limited to work with simple, regular grids. Other schemes can be provided independently. This is interesting when special regridding-needs arise or when more involved grids need to be considered. Furthermore, it may be desirable to have finer control over the parameters of the scheme than is afforded by the built-in schemes described above.

To facilitate this, the `regrid()` preprocessor allows the use of any scheme designed for Iris. The scheme must be installed and importable. Several such schemes are provided by `iris.analysis` and `esmvalcore.preprocessor.regrid_schemes`. To use this feature, the `scheme` key passed to the preprocessor must be a dictionary instead of a simple string that contains all necessary information. That includes a `reference` to the desired scheme itself, as well as any arguments that should be passed through to the scheme. For example, the following shows the use of the built-in scheme `iris.analysis.AreaWeighted` with a custom threshold for missing data tolerance.

```
preprocessors:
  regrid_preprocessor:
    regrid:
      target_grid: 2.5x2.5
      scheme:
        reference: iris.analysis:AreaWeighted
        mdtol: 0.7
```

Another example is bilinear regridding with extrapolation. This can be achieved with the `iris.analysis.Linear` scheme and the `extrapolation_mode` keyword. Extrapolation points will be calculated by extending the gradient of the closest two points.

```
preprocessors:
  regrid_preprocessor:
    regrid:
      target_grid: 2.5x2.5
      scheme:
        reference: iris.analysis:Linear
        extrapolation_mode: extrapolate
```

Note

Controlling the extrapolation mode allows us to avoid situations where extrapolating values makes little physical sense (e.g. extrapolating beyond the last data point).

The value of the `reference` key has two parts that are separated by a `:` with no surrounding spaces. The first part is an importable Python module, the second refers to the scheme, i.e. some callable that will be called with the remaining entries of the `scheme` dictionary passed as keyword arguments.

One package that aims to capitalize on the [support for meshes introduced in Iris 3.2](#) is `iris-esmf-regrid`. It aims to provide lazy regridding for structured regular and irregular grids, as well as meshes. It is recommended to use these schemes through the `esmvalcore.preprocessor.regrid_schemes.IrisESMFRegrid` scheme though, as that provides more efficient handling of masks.

An example of its usage in a preprocessor is:

```
preprocessors:
  regrid_preprocessor:
```

(continues on next page)

(continued from previous page)

```

regrid:
  target_grid: 2.5x2.5
  scheme:
    reference: esmvalcore.preprocessor.regrid_schemes:IrisESMFRegrid
    method: conservative
    mdtol: 0.7
    use_src_mask: true
    collapse_src_mask_along: ZT

```

Additionally, the use of generic schemes that take source and target grid cubes as arguments is also supported. The call function for such schemes must be defined as *(src_cube, grid_cube, **kwargs)* and they must return *iris.cube.Cube* objects. The *regrid* module will automatically pass the source and grid cubes as inputs of the scheme. An example of this usage is the `regrid_rectilinear_to_rectilinear()` scheme available in *iris-esmf-regrid*:

```

preprocessors:
  regrid_preprocessor:
    regrid:
      target_grid: 2.5x2.5
      scheme:
        reference: esmf_regrid.schemes:regrid_rectilinear_to_rectilinear
        mdtol: 0.7

```

Reusing regridding weights

If desired, regridding weights can be cached to reduce run times (see [here](#) for technical details on this). This can speed up the regridding of different datasets with similar source and target grids massively, but may take up a lot of memory for extremely high-resolution data. By default, this feature is disabled; to enable it, use the option `cache_weights: true` in the preprocessor definition:

```

preprocessors:
  regrid_preprocessor:
    regrid:
      target_grid: 0.1x0.1
      scheme: linear
      cache_weights: true

```

Not all regridding schemes support weights caching. An overview of those that do is given [here](#) and in the docstrings [here](#).

See also `esmvalcore.preprocessor.regrid()`

3.2.10 Ensemble statistics

For certain use cases it may be desirable to compute ensemble statistics. For example to prevent models with many ensemble members getting excessive weight in the multi-model statistics functions.

Theoretically, ensemble statistics are a special case (grouped) multi-model statistics. This grouping is performed taking into account the dataset tags *project*, *dataset*, *experiment*, and (if present) *sub_experiment*. However, they should typically be computed earlier in the workflow. Moreover, because multiple ensemble members of the same model are typically more consistent/homogeneous than datasets from different models, the implementation is more straightforward and can benefit from lazy evaluation and more efficient computation.

The preprocessor takes a list of statistics as input:

```
preprocessors:
  example_preprocessor:
    ensemble_statistics:
      statistics: [mean, median]
```

Additional keyword arguments can be given by using a dictionary:

```
preprocessors:
  example_preprocessor:
    ensemble_statistics:
      statistics:
        - operator: percentile
          percent: 20
        - operator: median
```

This preprocessor function exposes the iris analysis package, and works with all (capitalized) statistics from the `iris.analysis` package that can be executed without additional arguments. See *Statistical preprocessors* for more details on supported statistics.

Note that `ensemble_statistics` will not return the single model and ensemble files, only the requested ensemble statistics results.

In case of wanting to save both individual ensemble members as well as the statistic results, the preprocessor chains could be defined as:

```
preprocessors:
  everything_else: &everything_else
  area_statistics: ...
  regrid_time: ...
multimodel:
  <<: *everything_else
  ensemble_statistics:

variables:
  tas_datasets:
    short_name: tas
    preprocessor: everything_else
    ...
  tas_multimodel:
    short_name: tas
    preprocessor: multimodel
    ...
```

See also `esmvalcore.preprocessor.ensemble_statistics()`.

3.2.11 Multi-model statistics

Computing multi-model statistics is an integral part of model analysis and evaluation: individual models display a variety of biases depending on model set-up, initial conditions, forcings and implementation; comparing model data to observational data, these biases have a significantly lower statistical impact when using a multi-model ensemble. ESMValCore has the capability of computing a number of multi-model statistical measures: using the preprocessor module `multi_model_statistics` will enable the user for example to ask for either a multi-model mean, median, max, min, std_dev, and / or percentile with a set of argument parameters passed to `multi_model_statistics`. See *Statistical preprocessors* for more details on supported statistics. Percentiles can be specified with additional keyword arguments using the syntax `statistics: [{operator: percentile, percent: xx}]`.

Restrictive computation is also available by excluding any set of models that the user will not want to include in the statistics (by setting `exclude: [excluded models list]` argument).

Input datasets may have different time coordinates. Apart from that, all dimensions must match. Statistics can be computed across overlapping times only (`span: overlap`) or across the full time span of the combined models (`span: full`). The preprocessor sets a common time coordinate on all datasets. As the number of days in a year may vary between calendars, (sub-)daily data with different calendars are not supported. The preprocessor saves both the input single model files as well as the multi-model results. In case you do not want to keep the single model files, set the parameter `keep_input_datasets` to `false` (default value is `true`). To remove scalar coordinates before merging input datasets into the multi-dataset cube, use the option `ignore_scalar_coords: true`. The resulting multi-dataset cube will not have scalar coordinates in this case. This ensures that differences in scalar coordinates in the input datasets are ignored, which is helpful if you encounter a `ValueError: Multi-model statistics failed to merge input cubes into a single array with Coordinates in cube.aux_coords (scalar) differ`. Some special scalar coordinates which are expected to differ across cubes (*p0* and *ptop*) are always removed.

```
preprocessors:
  multi_model_save_input:
    multi_model_statistics:
      span: overlap
      statistics: [mean, median]
      exclude: [NCEP-NCAR-R1]
  multi_model_without_saving_input:
    multi_model_statistics:
      span: overlap
      statistics: [mean, median]
      exclude: [NCEP-NCAR-R1]
      keep_input_datasets: false
      ignore_scalar_coords: true
  multi_model_percentiles_5_95:
    multi_model_statistics:
      span: overlap
      statistics:
        - operator: percentile
          percent: 5
        - operator: percentile
          percent: 95
```

Multi-model statistics also supports a `groupby` argument. You can group by any dataset key (`project`, `experiment`, etc.) or a combination of keys in a list. You can also add an arbitrary tag to a dataset definition and then group by that tag. When using this preprocessor in conjunction with *ensemble statistics* preprocessor, you can group by `ensemble_statistics` as well. For example:

```
datasets:
- {dataset: CanESM2, exp: historical, ensemble: "r(1:2)i1p1"}
- {dataset: CCSM4, exp: historical, ensemble: "r(1:2)i1p1"}

preprocessors:
  example_preprocessor:
    ensemble_statistics:
      statistics: [median, mean]
    multi_model_statistics:
      span: overlap
      statistics: [min, max]
      groupby: [ensemble_statistics]
```

(continues on next page)

(continued from previous page)

```
exclude: [NCEP-NCAR-R1]
```

This will first compute ensemble mean and median, and then compute the multi-model min and max separately for the ensemble means and medians. Note that this combination will not save the individual ensemble members, only the ensemble and multimodel statistics results.

When grouping by a tag not defined in all datasets, the datasets missing the tag will be grouped together. In the example below, datasets *UKESM* and *ERA5* would belong to the same group, while the other datasets would belong to either group1 or group2

```
datasets:
- {dataset: CanESM2, exp: historical, ensemble: "r(1:2)i1p1", tag: 'group1'}
- {dataset: CanESM5, exp: historical, ensemble: "r(1:2)i1p1", tag: 'group2'}
- {dataset: CCSM4, exp: historical, ensemble: "r(1:2)i1p1", tag: 'group2'}
- {dataset: UKESM, exp: historical, ensemble: "r(1:2)i1p1"}
- {dataset: ERA5}

preprocessors:
example_preprocessor:
multi_model_statistics:
span: overlap
statistics: [min, max]
groupby: [tag]
```

Note that those datasets can be excluded if listed in the `exclude` option.

See also `esmvalcore.preprocessor.multi_model_statistics()`.

3.2.12 Time manipulation

The `_time.py` module contains the following preprocessor functions:

- `extract_time`: Extract a time range from a cube.
- `extract_season`: Extract only the times that occur within a specific season.
- `extract_month`: Extract only the times that occur within a specific month.
- `hourly_statistics`: Compute intra-day statistics
- `daily_statistics`: Compute statistics for each day
- `monthly_statistics`: Compute statistics for each month
- `seasonal_statistics`: Compute statistics for each season
- `annual_statistics`: Compute statistics for each year
- `decadal_statistics`: Compute statistics for each decade
- `climate_statistics`: Compute statistics for the full period
- `resample_time`: Resample data
- `resample_hours`: Convert between N-hourly frequencies by resampling
- `anomalies`: Compute (standardized or relative) anomalies
- `regrid_time`: Aligns the time coordinate of each dataset, against a standardized time axis.
- `timeseries_filter`: Allows application of a filter to the time-series data.

- *local_solar_time*: Convert cube with UTC time to local solar time.

Statistics functions are applied by default in the order they appear in the list. For example, the following example applied to hourly data will retrieve the minimum values for the full period (by season) of the monthly mean of the daily maximum of any given variable.

```
daily_statistics:  
  operator: max  
  
monthly_statistics:  
  operator: mean  
  
climate_statistics:  
  operator: min  
  period: season
```

`extract_time`

This function extracts data within specific time criteria. The preprocessor removes all times which fall outside the specified time range. The required arguments are relatively self explanatory:

- `start_year`
- `start_month`
- `start_day`
- `end_year`
- `end_month`
- `end_day`

The start and end points are set using the datasets native calendar. `start_month`, `start_day`, `end_month`, and `end_day` should be given as integers - the named month string will not be accepted. `start_year` and `end_year` should both be either integers or null. If `start_year` and `end_year` are null, the date ranges (`start_month`-`start_day` to `end_month`-`end_day`) are selected in each year. For example, ranges Feb 3 - Apr 6 in each year are selected with the following preprocessor:

```
extract_time:  
  start_year: null  
  start_month: 2  
  start_day: 3  
  end_year: null  
  end_month: 4  
  end_day: 6
```

And the period between Feb 3, 2001 - Apr 6, 2004 is selected as follows:

```
extract_time:  
  start_year: 2001  
  start_month: 2  
  start_day: 3  
  end_year: 2004  
  end_month: 4  
  end_day: 6
```

See also `esmvalcore.preprocessor.extract_time()`.

extract_season

Extract only the times that occur within a specific season.

This function only has one argument: `season`. This is the named season to extract, i.e. DJF, MAM, JJA, SON, but also all other sequentially correct combinations, e.g. JJAS.

Note that this function does not change the time resolution. If your original data is in monthly time resolution, then this function will return three monthly datapoints per year.

If you want the seasonal average, then this function needs to be combined with the `seasonal_mean` function, below.

See also `esmvalcore.preprocessor.extract_season()`.

extract_month

The function extracts the times that occur within a specific month. This function only has one argument: `month`. This value should be an integer between 1 and 12 as the named month string will not be accepted.

See also `esmvalcore.preprocessor.extract_month()`.

hourly_statistics

This function produces statistics at a x-hourly frequency.

Parameters:

- `hours`: Number of hours per period. Must be a divisor of 24, i.e., (1, 2, 3, 4, 6, 8, 12).
- `operator`: Operation to apply. See *Statistical preprocessors* for more details on supported statistics. Default is *mean*.
- Other parameters are directly passed to the `operator` as keyword arguments. See *Statistical preprocessors* for more details.

See also `esmvalcore.preprocessor.hourly_statistics()`.

daily_statistics

This function produces statistics for each day in the dataset.

Parameters:

- `operator`: Operation to apply. See *Statistical preprocessors* for more details on supported statistics. Default is *mean*.
- Other parameters are directly passed to the `operator` as keyword arguments. See *Statistical preprocessors* for more details.

See also `esmvalcore.preprocessor.daily_statistics()`.

monthly_statistics

This function produces statistics for each month in the dataset.

Parameters:

- `operator`: Operation to apply. See *Statistical preprocessors* for more details on supported statistics. Default is *mean*.
- Other parameters are directly passed to the `operator` as keyword arguments. See *Statistical preprocessors* for more details.

See also `esmvalcore.preprocessor.monthly_statistics()`.

seasonal_statistics

This function produces statistics for each season (default: [DJF, MAM, JJA, SON] or custom seasons e.g. [JJAS, ONDJFMAM]) in the dataset. Note that this function will not check for missing time points. For instance, if you are looking at the DJF field, but your datasets starts on January 1st, the first DJF field will only contain data from January and February.

We recommend using the `extract_time` to start the dataset from the following December and remove such biased initial data points.

Parameters:

- *operator*: Operation to apply. See *Statistical preprocessors* for more details on supported statistics. Default is *mean*.
- *seasons*: Seasons to build statistics. Default is '[DJF, MAM, JJA, SON]'.
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

See also `esmvalcore.preprocessor.seasonal_statistics()`.

annual_statistics

This function produces statistics for each year.

Parameters:

- *operator*: Operation to apply. See *Statistical preprocessors* for more details on supported statistics. Default is *mean*.
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

See also `esmvalcore.preprocessor.annual_statistics()`.

decadal_statistics

This function produces statistics for each decade.

Parameters:

- *operator*: Operation to apply. See *Statistical preprocessors* for more details on supported statistics. Default is *mean*.
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

See also `esmvalcore.preprocessor.decadal_statistics()`.

climate_statistics

This function produces statistics for the whole dataset. It can produce scalars (if the full period is chosen) or hourly, daily, monthly or seasonal statistics.

Parameters:

- *operator*: Operation to apply. See *Statistical preprocessors* for more details on supported statistics. Default is *mean*.
- *period*: Define the granularity of the statistics: get values for the full period, for each month, day of year or hour of day. Available periods: *full, season, seasonal, monthly, month, mon, daily, day, hourly, hour, hr*. Default is *full*.

- *seasons*: if period 'seasonal' or 'season' allows to set custom seasons. Default is '[DJF, MAM, JJA, SON]'
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

Note

Some operations are weighted by the time coordinate by default, i.e., the length of the time intervals. See *Statistical preprocessors* for more details on supported statistics. For *sum*, the units of the resulting cube are multiplied by the corresponding time units (e.g., days).

If a period other than *full* is used, time points will be put into bins, which may shift existing time points. For example, for *period=hourly*, a time point at 01:30h will be moved to the corresponding full hour (01:00h).

Examples:

- Monthly climatology:

```
climate_statistics:
  operator: mean
  period: month
```

- Daily maximum for the full period:

```
climate_statistics:
  operator: max
  period: day
```

- Minimum value in the period:

```
climate_statistics:
  operator: min
  period: full
```

- 80% percentile for each month:

```
climate_statistics:
  period: month
  operator: percentile
  percent: 80
```

See also `esmvalcore.preprocessor.climate_statistics()`.

resample_time

This function changes the frequency of the data in the cube by extracting the timesteps that meet the criteria. It is important to note that it is mainly meant to be used with instantaneous data.

Parameters:

- *month*: Extract only timesteps from the given month or do nothing if *None*. Default is *None*
- *day*: Extract only timesteps from the given day of month or do nothing if *None*. Default is *None*
- *hour*: Extract only timesteps from the given hour or do nothing if *None*. Default is *None*

Examples:

- Hourly data to daily:

```
resample_time:  
hour: 12
```

- Hourly data to monthly:

```
resample_time:  
hour: 12  
day: 15
```

- Daily data to monthly:

```
resample_time:  
day: 15
```

See also `esmvalcore.preprocessor.resample_time()`.

`resample_hours`:

`resample_hours`

Change the frequency of x-hourly data to y-hourly data by either eliminating extra time steps or interpolation. This is intended to be used with instantaneous data.

Parameters:

- *interval* (`int`): New frequency of the data. Must be a divisor of 24.
- *offset* (`int`): First hour of the desired output data (default: 0). Must be lower than the value of *interval*.
- *interpolate* (`None` or `str`): If *interpolate* is `None` (default), convert x-hourly data to y-hourly ($y > x$) by eliminating extra time steps. If *interpolate* is 'nearest' or 'linear', use nearest-neighbor or bilinear interpolation to convert general x-hourly data to general y-hourly data.

Examples:

- Convert to 12-hourly data by getting time steps at 0:00 and 12:00:

```
resample_hours:  
interval: 12
```

- Convert to 12-hourly data by getting time steps at 6:00 and 18:00:

```
resample_hours:  
interval: 12  
offset: 6
```

- Convert to 3-hourly data using bilinear interpolation:

```
resample_hours:  
interval: 3  
interpolate: linear
```

See also `esmvalcore.preprocessor.resample_hours()`.

anomalies

This function computes the anomalies for the whole dataset. It can compute anomalies from the full, seasonal, monthly, daily and hourly climatologies. Optionally standardized anomalies can be calculated.

Parameters:

- `period`: define the granularity of the climatology to use: full period, seasonal, monthly, daily or hourly. Available periods: 'full', 'season', 'seasonal', 'monthly', 'month', 'mon', 'daily', 'day', 'hourly', 'hour', 'hr'. Default is 'full'
- `reference`: Time slice to use as the reference to compute the climatology on. Can be 'null' to use the full cube or a dictionary with the parameters from *extract_time*. Default is null
- `relative`: if true, calculate relative (in percent) anomalies (default: false)
- `standardize`: if true calculate standardized anomalies (default: false)
- `seasons`: if period 'seasonal' or 'season' allows to set custom seasons. Default is '[DJF, MAM, JJA, SON]'

Examples:

- Anomalies from the full period climatology:

```
anomalies:
```

- Anomalies from the full period monthly climatology:

```
anomalies:
  period: month
```

- Standardized anomalies from the full period climatology:

```
anomalies:
  standardized: true
```

- Standardized Anomalies from the 1979-2000 monthly climatology:

```
anomalies:
  period: month
  reference:
    start_year: 1979
    start_month: 1
    start_day: 1
    end_year: 2000
    end_month: 12
    end_day: 31
  standardize: true
```

See also *esmvalcore.preprocessor.anomalies()*.

regrid_time

This function aligns the time points and bounds of an input dataset according to the following rules:

- Decadal data: 1 January 00:00:00 for the given year. Example: 1 January 2005 00:00:00 for given year 2005 (decade 2000-2010).
- Yearly data: 1 July 00:00:00 for each year. Example: 1 July 1993 00:00:00 for the year 1993.

- Monthly data: 15th day 00:00:00 for each month. Example: 15 October 1993 00:00:00 for the month October 1993.
- Daily data: 12:00:00 for each day. Example: 14 March 1996 12:00:00 for the day 14 March 1996.
- n -hourly data where n is a divisor of 24: center of each time interval. Example: 03:00:00 for interval 00:00:00-06:00:00 (6-hourly data), 16:30:00 for interval 15:00:00-18:00:00 (3-hourly data), or 09:30:00 for interval 09:00:00-10:00:00 (hourly data).

The frequency of the input data is automatically determined from the CMOR table of the corresponding variable, but can be overwritten in the recipe if necessary. This function does not alter the data in any way.

Note

By default, this preprocessor will not change the calendar of the input time coordinate. For decadal, yearly, and monthly data, it is possible to change the calendar using the optional *calendar* argument. Be aware that changing the calendar might introduce (small) errors to your data, especially for extensive quantities (those that depend on the period length).

Parameters:

- *frequency*: Data frequency. If not given, use the one from the CMOR tables of the corresponding variable.
- *calendar*: If given, transform the calendar to the one specified (examples: *standard*, *365_day*, etc.). This only works for decadal, yearly and monthly data, and will raise an error for other frequencies. If not set, the calendar will not be changed.
- *units* (default: *days since 1850-01-01 00:00:00*): Reference time units used if the calendar of the data is changed. Ignored if *calendar* is not set.

Examples:

Change the input calendar to *standard* and use custom units:

```
regrid_time:
  calendar: standard
  units: days since 2000-01-01
```

See also `esmvalcore.preprocessor.regrid_time()`.

timeseries_filter

This function allows the user to apply a filter to the timeseries data. This filter may be of the user's choice (currently only the low-pass Lanczos filter is implemented); the implementation is inspired by this [iris example](#) and uses aggregation via `iris.cube.Cube.rolling_window`.

Parameters:

- *window*: the length of the filter window (in units of cube time coordinate).
- *span*: period (number of months/days, depending on data frequency) on which weights should be computed e.g. for 2-yearly: $\text{span} = 24$ (2 x 12 months). Make sure span has the same units as the data cube time coordinate.
- *filter_type*: the type of filter to be applied; default 'lowpass'. Available types: 'lowpass'.
- *filter_stats*: the type of statistic to aggregate on the rolling window; default 'sum'. Available operators: 'mean', 'median', 'std_dev', 'sum', 'min', 'max', 'rms'.

Examples:

- Lowpass filter with a monthly mean as operator:

```

timeseries_filter:
  window: 3 # 3-monthly filter window
  span: 12 # weights computed on the first year
  filter_type: lowpass # low-pass filter
  filter_stats: mean # 3-monthly mean lowpass filter

```

See also `esmvalcore.preprocessor.timeseries_filter()`.

local_solar_time

Many variables in the Earth system show a strong diurnal cycle. The reason for that is of course Earth's rotation around its own axis, which leads to a diurnal cycle of the incoming solar radiation. While UTC time is a very good absolute time measure, it is not really suited to analyze diurnal cycles over larger regions. For example, diurnal cycles over Russia and the USA are phase-shifted by $\sim 180^\circ = 12$ hr in UTC time.

This is where the **local solar time (LST)** comes into play: For a given location, 12:00 noon LST is defined as the moment when the sun reaches its highest point in the sky. By using this definition based on the origin of the diurnal cycle (the sun), we can directly compare diurnal cycles across the globe. LST is mainly determined by the longitude of a location, but due to the eccentricity of Earth's orbit, it also depends on the day of year (see [equation of time](#)). However, this correction is at most ~ 15 min, which is usually smaller than the highest frequency output of CMIP6 models (1 hr) and smaller than the time scale for diurnal evolution of meteorological phenomena (which is in the order of hours, not minutes). Thus, instead, we use the **mean LST**, which solely depends on longitude:

$$LST = UTC + 12 \cdot \frac{lon}{180}$$

where the times are given in hours and *lon* in degrees in the interval $[-180, 180]$. To transform data from UTC to LST, this preprocessor shifts data along the time axis based on the longitude.

This preprocessor does not need any additional parameters.

Example:

```

calculate_local_solar_time:
  local_solar_time:

```

See also `esmvalcore.preprocessor.local_solar_time()`.

3.2.13 Area manipulation

The area manipulation module contains the following preprocessor functions:

- `extract_coordinate_points`: Extract a point with arbitrary coordinates given an interpolation scheme.
- `extract_region`: Extract a region from a cube based on lat/lon corners.
- `extract_named_regions`: Extract a specific region from in the region coordinate.
- `extract_shape`: Extract a region defined by a shapefile.
- `extract_point`: Extract a single point (with interpolation)
- `extract_location`: Extract a single point by its location (with interpolation)
- `zonal_statistics`: Compute zonal statistics.
- `meridional_statistics`: Compute meridional statistics.
- `area_statistics`: Compute area statistics.

extract_coordinate_points

This function extracts points with given coordinates, following either a `linear` or a `nearest` interpolation scheme. The resulting point cube will match the respective coordinates to those of the input coordinates. If the input coordinate is a scalar, the dimension will be a scalar in the output cube.

If the point to be extracted has at least one of the coordinate point values outside the interval of the cube's same coordinate values, then no extrapolation will be performed, and the resulting extracted cube will have fully masked data.

Examples:

- Extract a point from coordinate `grid_latitude` with given coordinate value 26.0:

```
extract_coordinate_points:  
  definition:  
    grid_latitude: 26.  
  scheme: nearest
```

See also `esmvalcore.preprocessor.extract_coordinate_points()`.

extract_region

This function returns a subset of the data on the rectangular region requested. The boundaries of the region are provided as latitude and longitude coordinates in the arguments:

- `start_longitude`
- `end_longitude`
- `start_latitude`
- `end_latitude`

Note that this function can only be used to extract a rectangular region. Use `extract_shape` to extract any other shaped region from a shapefile.

If the grid is irregular, the returned region retains the original coordinates, but is cropped to a rectangular bounding box defined by the start/end coordinates. The deselected area inside the region is masked.

See also `esmvalcore.preprocessor.extract_region()`.

extract_named_regions

This function extracts a specific named region from the data. This function takes the following argument: `regions` which is either a string or a list of strings of named regions. Note that the dataset must have a `region` coordinate which includes a list of strings as values. This function then matches the named regions against the requested string.

See also `esmvalcore.preprocessor.extract_named_regions()`.

extract_shape

Extract a shape or a representative point for this shape from the data.

Parameters:

- `shapefile`: path to the shapefile containing the geometry of the region to be extracted. If the file contains multiple shapes behaviour depends on the `decomposed` parameter. This path can be relative to the directory specified via the *configuration option* `auxiliary_data_dir` or relative to `esmvalcore/preprocessor/shapefiles` (in that priority order). Alternatively, a string (see “Shapefile name” below) can be given to load one of the following shapefiles that are shipped with ESMValCore:

Shapefile name	Description	Reference
ar6	IPCC WG1 reference regions (v4) used in Assessment Report 6	https://doi.org/10.5281/zenodo.5176260

- **method**: the method to select the region, selecting either all points contained by the shape or a single representative point. Choose either *'contains'* or *'representative'*. If not a single grid point is contained in the shape, a representative point will be selected.
- **crop**: by default *extract_region* will be used to crop the data to a minimal rectangular region containing the shape. Set to *false* to only mask data outside the shape. Data on irregular grids will not be cropped.
- **decomposed**: by default *false*; in this case the union of all the regions in the shapefile is masked out. If set to *true*, the regions in the shapefiles are masked out separately and the output cube will have an additional dimension *shape_id* describing the requested regions.
- **ids**: Shapes to be read from the shapefile. Can be given as:
 - **list**: IDs are assigned from the attributes *name*, *NAME*, *Name*, *id*, or *ID* (in that priority order; the first one available is used). If none of these attributes are available in the shapefile, assume that the given *ids* correspond to the reading order of the individual shapes. So, for example, if a file has both *name* and *id* attributes, the *ids* will be assigned from *name*. If the file only has the *id* attribute, it will be taken from it and if no *name* nor *id* attributes are present, an integer ID starting from 0 will be assigned automatically when reading the shapes. We discourage to rely on this last behaviour as we can not assure that the reading order will be the same on different platforms, so we encourage you to specify a custom attribute using a **dict** (see below) instead. Note: An empty list is interpreted as *ids=None* (see below).
 - **dict**: IDs (dictionary value; **list** of **str**) are assigned from attribute given as dictionary key (**str**). Only dictionaries with length 1 are supported. Example: `ids={'Acronym': ['GIC', 'WNA']}`.
 - *None*: select all available regions from the shapefile.

Examples:

- Extract the shape of the river Elbe from a shapefile:

```
extract_shape:
  shapefile: Elbe.shp
  method: contains
```

- Extract the shape of several countries:

```
extract_shape:
  shapefile: NaturalEarth/Countries/ne_110m_admin_0_countries.shp
  decomposed: true
  method: contains
  ids:
    - Spain
    - France
    - Italy
    - United Kingdom
    - Taiwan
```

- Extract European AR6 regions:

```
extract_shape:
  shapefile: ar6
  method: contains
  ids:
    Acronym:
      - NEU
      - WCE
      - MED
```

See also `esmvalcore.preprocessor.extract_shape()`.

extract_point

Extract a single point from the data. This is done using either nearest or linear interpolation.

Returns a cube with the extracted point(s), and with adjusted latitude and longitude coordinates (see below).

Multiple points can also be extracted, by supplying an array of latitude and/or longitude coordinates. The resulting point cube will match the respective latitude and longitude coordinate to those of the input coordinates. If the input coordinate is a scalar, the dimension will be missing in the output cube (that is, it will be a scalar).

If the point to be extracted has at least one of the coordinate point values outside the interval of the cube's same coordinate values, then no extrapolation will be performed, and the resulting extracted cube will have fully masked data.

Parameters:

- `cube`: the input dataset cube.
- `latitude`, `longitude`: coordinates (as floating point values) of the point to be extracted. Either (or both) can also be an array of floating point values.
- `scheme`: interpolation scheme: either 'linear' or 'nearest'. There is no default.

See also `esmvalcore.preprocessor.extract_point()`.

extract_location

Extract a single point using a location name, with interpolation (either linear or nearest). This preprocessor extracts a single location point from a cube, according to the given interpolation scheme `scheme`. The function retrieves the coordinates of the location and then calls the `esmvalcore.preprocessor.extract_point()` preprocessor. It can be used to locate cities and villages, but also mountains or other geographical locations.

Note

Note that this function's geolocator application needs a working internet connection.

Parameters:

- `cube`: the input dataset cube to extract a point from.
- `location`: the reference location. Examples: 'mount everest', 'romania', 'new york, usa'. Raises `ValueError` if none supplied.
- `scheme`: interpolation scheme. *linear* or *nearest*. There is no default, raises `ValueError` if none supplied.

See also `esmvalcore.preprocessor.extract_location()`.

zonal_statistics

The function calculates the zonal statistics by applying an operator along the longitude coordinate.

Parameters:

- *operator*: Operation to apply. See *Statistical preprocessors* for more details on supported statistics.
- *normalize*: If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be *subtract* (statistics cube is subtracted from the input cube) or *divide* (input cube is divided by the statistics cube).
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

See also `esmvalcore.preprocessor.zonal_statistics()`.

meridional_statistics

The function calculates the meridional statistics by applying an operator along the latitude coordinate. This function takes one argument:

Parameters:

- *operator*: Operation to apply. See *Statistical preprocessors* for more details on supported statistics.
- *normalize*: If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be *subtract* (statistics cube is subtracted from the input cube) or *divide* (input cube is divided by the statistics cube).
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

See also `esmvalcore.preprocessor.meridional_statistics()`.

area_statistics

This function can be used to apply several different operations in the horizontal plane: for example, mean, sum, standard deviation, median, variance, minimum, maximum and root mean square. Some operations are grid cell area weighted by default. For sums, the units of the resulting cubes are multiplied by m^2 . See *Statistical preprocessors* for more details on supported statistics.

Note that this function is applied over the entire dataset. If only a specific region, depth layer or time period is required, then those regions need to be removed using other preprocessor operations in advance.

For weighted statistics, this function requires a cell area `cell measure`, unless the coordinates of the input data are regular 1D latitude and longitude coordinates so the cell areas can be computed internally. The required supplementary variable, either `areacella` for atmospheric variables or `areacello` for ocean variables, can be attached to the main dataset as described in *Defining supplementary variables (ancillary variables and cell measures)*.

Parameters:

- *operator*: Operation to apply. See *Statistical preprocessors* for more details on supported statistics.
- *normalize*: If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be *subtract* (statistics cube is subtracted from the input cube) or *divide* (input cube is divided by the statistics cube).
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

Examples: * Calculate global mean:

```
area_statistics:  
  operator: mean
```

- Subtract global mean from dataset:

```
area_statistics:  
  operator: mean  
  normalize: subtract
```

See also `esmvalcore.preprocessor.area_statistics()`.

3.2.14 Volume manipulation

The `_volume.py` module contains the following preprocessor functions:

- `axis_statistics`: Perform operations along a given axis.
- `extract_volume`: Extract a specific depth range from a cube.
- `volume_statistics`: Calculate the volume-weighted average.
- `depth_integration`: Integrate over the depth dimension.
- `extract_transect`: Extract data along a line of constant latitude or longitude.
- `extract_trajectory`: Extract data along a specified trajectory.
- `extract_surface_from_atm`: Extract atmospheric data at the surface.

`extract_volume`

Extract a specific range in the z -direction from a cube. The range is given as an interval that can be:

- open (z_{\min}, z_{\max}) , in which the extracted range does not include z_{\min} nor z_{\max} .
- closed $[z_{\min}, z_{\max}]$, in which the extracted includes both z_{\min} and z_{\max} .
- left closed $[z_{\min}, z_{\max})$, in which the extracted range includes z_{\min} but not z_{\max} .
- right closed $(z_{\min}, z_{\max}]$, in which the extracted range includes z_{\max} but not z_{\min} .

The extraction is performed by applying a constraint on the coordinate values, without any kind of interpolation.

This function takes four arguments:

- `z_min` to define the minimum value of the range to extract in the z -direction.
- `z_max` to define the maximum value of the range to extract in the z -direction.
- **interval_bounds to define whether the bounds of the interval are open, closed, left_closed or right_closed.** Default is open.
- **nearest_value to extract a range taking into account the values of the z -coordinate that are closest to z_{\min} and z_{\max} .** Default is `False`.

As the coordinate points are likely to vary depending on the dataset, sometimes it might be useful to adjust the given `z_min` and `z_max` values to the values of the coordinate points before performing an extraction. This behaviour can be achieved by setting the `nearest_value` argument to `True`.

For example, in a cube with `z_coord = [0., 1.5, 2.6., 3.8., 5.4]`, the preprocessor below:

```
preprocessors:
  extract_volume:
    z_min: 1.
    z_max: 5.
    interval_bounds: 'closed'
```

would return a cube with a `z_coord` defined as `z_coord = [1.5, 2.6., 3.8.]`, since these are the values that strictly fall into the range given by `[z_min=1, z_max=5]`.

Whereas setting `nearest_value: True`:

```
preprocessors:
  extract_volume:
    z_min: 1.
    z_max: 5.
    interval_bounds: 'closed'
    nearest_value: True
```

would return a cube with a `z_coord` defined as `z_coord = [1.5, 2.6., 3.8., 5.4]`, since `z_max = 5` is closest to the coordinate point `z = 5.4` than it is to `z = 3.8`.

Note that this preprocessor requires the requested `z`-coordinate range to be the same sign as the Iris cube. That is, if the cube has `z`-coordinate as negative, then `z_min` and `z_max` need to be negative numbers.

See also `esmvalcore.preprocessor.extract_volume()`.

volume_statistics

This function calculates the volume-weighted average across three dimensions, but maintains the time dimension.

By default, the *mean* operation is weighted by the grid cell volumes.

For weighted statistics, this function requires a cell volume `cell measure`, unless it has a `cell_area` `cell measure` or the coordinates of the input data are regular 1D latitude and longitude coordinates so the cell volumes can be computed internally. The required supplementary variable `volcello`, or `areacello` in its absence, can be attached to the main dataset as described in *Defining supplementary variables (ancillary variables and cell measures)*.

No depth coordinate is required as this is determined by Iris. However, to compute the volume automatically when `volcello` is not provided, the depth coordinate units should be convertible to meters.

Parameters:

- *operator*: Operation to apply. At the moment, only *mean* is supported. See *Statistical preprocessors* for more details on supported statistics.
- *normalize*: If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be *subtract* (statistics cube is subtracted from the input cube) or *divide* (input cube is divided by the statistics cube).
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

See also `esmvalcore.preprocessor.volume_statistics()`.

axis_statistics

This function operates over a given axis, and removes it from the output cube.

Takes arguments:

- *axis*: direction over which the statistics will be performed. Possible values for the axis are *x*, *y*, *z*, *t*.

- *operator*: Operation to apply. See *Statistical preprocessors* for more details on supported statistics.
- *normalize*: If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be *subtract* (statistics cube is subtracted from the input cube) or *divide* (input cube is divided by the statistics cube).
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

Note

The coordinate associated to the axis over which the operation will be performed must be one-dimensional, as multidimensional coordinates are not supported in this preprocessor.

Some operations are weighted by the corresponding coordinate bounds by default. For sums, the units of the resulting cubes are multiplied by the corresponding coordinate units. See *Statistical preprocessors* for more details on supported statistics.

See also `esmvalcore.preprocessor.axis_statistics()`.

depth_integration

This function integrates over the depth dimension. This function does a weighted sum along the *z*-coordinate, and removes the *z* direction of the output cube. This preprocessor takes no arguments. The units of the resulting cube are multiplied by the *z*-coordinate units.

See also `esmvalcore.preprocessor.depth_integration()`.

extract_transect

This function extracts data along a line of constant latitude or longitude. This function takes two arguments, although only one is strictly required. The two arguments are `latitude` and `longitude`. One of these arguments needs to be set to a float, and the other can then be either ignored or set to a minimum or maximum value.

For example, if we set `latitude` to 0 N and leave `longitude` blank, it would produce a cube along the Equator. On the other hand, if we set `latitude` to 0 and then set `longitude` to `[40., 100.]` this will produce a transect of the Equator in the Indian Ocean.

See also `esmvalcore.preprocessor.extract_transect()`.

extract_trajectory

This function extract data along a specified trajectory. The three arguments are: `latitudes`, `longitudes` and number of point needed for extrapolation `number_points`.

If two points are provided, the `number_points` argument is used to set a the number of places to extract between the two end points.

If more than two points are provided, then `extract_trajectory` will produce a cube which has extrapolated the data of the cube to those points, and `number_points` is not needed.

Note that this function uses the expensive `interpolate` method from `Iris.analysis.trajectory`, but it may be necessary for irregular grids.

See also `esmvalcore.preprocessor.extract_trajectory()`.

`extract_surface_from_atm`

This function extracts data at the surface for an atmospheric variable.

The function returns the interpolated value of an input field at the corresponding surface pressure given by the surface air pressure `ps`.

The required supplementary surface air pressure `ps` is attached to the main dataset as described in *Defining supplementary variables (ancillary variables and cell measures)*.

See also `esmvalcore.preprocessor.extract_surface_from_atm()`.

3.2.15 Cycles

The `_cycles.py` module contains the following preprocessor functions:

- `amplitude`: Extract the peak-to-peak amplitude of a cycle aggregated over specified coordinates.

`amplitude`

This function extracts the peak-to-peak amplitude (maximum value minus minimum value) of a field aggregated over specified coordinates. Its only argument is `coords`, which can either be a single coordinate (given as `str`) or multiple coordinates (given as `list of str`). Usually, these coordinates refer to temporal categorised coordinates `iris.coord_categorisation` like `year`, `month`, `day of year`, etc. For example, to extract the amplitude of the annual cycle for every single year in the data, use `coords: year`; to extract the amplitude of the diurnal cycle for every single day in the data, use `coords: [year, day_of_year]`.

See also `esmvalcore.preprocessor.amplitude()`.

3.2.16 Trend

The trend module contains the following preprocessor functions:

- `linear_trend`: Calculate linear trend along a specified coordinate.
- `linear_trend_stderr`: Calculate standard error of linear trend along a specified coordinate.

`linear_trend`

This function calculates the linear trend of a dataset (defined as slope of an ordinary linear regression) along a specified coordinate. The only argument of this preprocessor is `coordinate` (given as `str`; default value is `'time'`).

See also `esmvalcore.preprocessor.linear_trend()`.

`linear_trend_stderr`

This function calculates the standard error of the linear trend of a dataset (defined as the standard error of the slope in an ordinary linear regression) along a specified coordinate. The only argument of this preprocessor is `coordinate` (given as `str`; default value is `'time'`). Note that the standard error is **not** identical to a confidence interval.

See also `esmvalcore.preprocessor.linear_trend_stderr()`.

3.2.17 Detrend

ESMValCore also supports detrending along any dimension using the preprocessor function `'detrend'`. This function has two parameters:

- `dimension`: dimension to apply detrend on. Default: `"time"`
- `method`: It can be `linear` or `constant`. Default: `linear`

If method is `linear`, `detrend` will calculate the linear trend along the selected axis and subtract it to the data. For example, this can be used to remove the linear trend caused by climate change on some variables if selected dimension is time.

If method is `constant`, `detrend` will compute the mean along that dimension and subtract it from the data

See also `esmvalcore.preprocessor.detrend()`.

3.2.18 Rolling window statistics

One can calculate rolling window statistics using the preprocessor function `rolling_window_statistics`. This function takes three parameters:

- *coordinate*: Coordinate over which the rolling-window statistics is calculated.
- *operator*: Operation to apply. See *Statistical preprocessors* for more details on supported statistics.
- *window_length*: size of the rolling window to use (number of points).
- Other parameters are directly passed to the *operator* as keyword arguments. See *Statistical preprocessors* for more details.

This example applied on daily precipitation data calculates two-day rolling precipitation sum.

```
preprocessors:
  preproc_rolling_window:
    rolling_window_statistics:
      coordinate: time
      operator: sum
      window_length: 2
```

See also `esmvalcore.preprocessor.rolling_window_statistics()`.

3.2.19 Unit conversion

`convert_units`

Converting units is also supported. This is particularly useful in cases where different datasets might have different units, for example when comparing CMIP5 and CMIP6 variables where the units have changed or in case of observational datasets that are delivered in different units.

In these cases, having a unit conversion at the end of the processing will guarantee homogeneous input for the diagnostics.

Conversion is only supported between compatible units! In other words, converting temperature units from `degC` to `Kelvin` works fine, while changing units from `kg` to `m` will not work.

However, there are some well-defined exceptions from this rule in order to transform one quantity to another (physically related) quantity. These quantities are identified via their `standard_name` and their `units` (units convertible to the ones defined are also supported). For example, this enables conversions between precipitation fluxes measured in `kg m-2 s-1` and precipitation rates measured in `mm day-1` (and vice versa). Currently, the following special conversions are supported:

- `precipitation_flux (kg m-2 s-1)` – `lwe_precipitation_rate (mm day-1)`
- `equivalent_thickness_at_stp_of_atmosphere_ozone_content (m)` – `equivalent_thickness_at_stp_of_atmosphere_ozone_content (DU)`

Hint

Names in the list correspond to `standard_names` of the input data. Conversions are allowed from each quantity to any other quantity given in a bullet point. The corresponding target quantity is inferred from the desired target units. In addition, any other units convertible to the ones given are also supported (e.g., instead of `mm day-1`, `m s-1` is also supported).

Note

For the transformation between the different precipitation variables, a water density of `1000 kg m-3` is assumed.

See also `esmvalcore.preprocessor.convert_units()`.

accumulate_coordinate

This function can be used to weight data using the bounds from a given coordinate. The resulting cube will then have units given by `cube_units * coordinate_units`.

For instance, if a variable has units such as `X s-1`, using `accumulate_coordinate` on the time coordinate would result on a cube where the data would be multiplied by the time bounds and the resulting units for the variable would be converted to `X`. In this case, weighting the data with the time coordinate would allow to cancel the time units in the variable.

Note

The coordinate used to weight the data must be one-dimensional, as multidimensional coordinates are not supported in this preprocessor.

See also `esmvalcore.preprocessor.accumulate_coordinate()`.

3.2.20 Comparison with reference dataset

This module contains the following preprocessor functions:

- `bias`: Calculate absolute or relative biases with respect to a reference dataset.
- `distance_metric`: Calculate absolute or relative biases with respect to a reference dataset.

bias

This function calculates biases with respect to a given reference dataset. For this, exactly one input dataset needs to be declared as `reference_for_bias: true` in the recipe, e.g.,

datasets:

- ```
- {dataset: CanESM5, project: CMIP6, ensemble: r1i1p1f1, grid: gn}
- {dataset: CESM2, project: CMIP6, ensemble: r1i1p1f1, grid: gn}
- {dataset: MIROC6, project: CMIP6, ensemble: r1i1p1f1, grid: gn}
- {dataset: ERA-Interim, project: OBS6, tier: 3, type: reanaly, version: 1,
 reference_for_bias: true}
```

In the example above, ERA-Interim is used as reference dataset for the bias calculation.

It is also possible to use the output from the *Multi-model statistics* or *Ensemble statistics* preprocessor as reference dataset. In this case, make sure to use `reference_for_bias: true` for each dataset that will be used to create the reference dataset and use the option `keep_input_datasets: false` for the multi-dataset preprocessor. For example:

```
datasets:
- {dataset: CanESM5, group: ref, reference_for_bias: true}
- {dataset: CESM2, group: ref, reference_for_bias: true}
- {dataset: MIROC6, group: notref}

preprocessors:
 calculate_bias:
 custom_order: true
 multi_model_statistics:
 statistics: [mean]
 span: overlap
 groupby: [group]
 keep_input_datasets: false
 bias:
 bias_type: relative
```

Here, the bias of MIROC6 is calculated relative to the multi-model mean from the models CanESM5 and CESM2.

The reference dataset needs to be broadcastable to all other datasets. This supports *iris'* rich broadcasting abilities. To ensure this, the preprocessors `esmvalcore.preprocessor.regrid()` and/or `esmvalcore.preprocessor.regrid_time()` might be helpful.

The bias preprocessor supports 4 optional arguments in the recipe:

- `bias_type` (`str`, default: 'absolute'): Bias type that is calculated. Can be 'absolute' (i.e., calculate bias for dataset  $X$  and reference  $R$  as  $X - R$ ) or relative (i.e., calculate bias as  $\frac{X-R}{R}$ ).
- `denominator_mask_threshold` (`float`, default: 1e-3): Threshold to mask values close to zero in the denominator (i.e., the reference dataset) during the calculation of relative biases. All values in the reference dataset with absolute value less than the given threshold are masked out. This setting is ignored when `bias_type` is set to 'absolute'. Please note that for some variables with very small absolute values (e.g., carbon cycle fluxes, which are usually  $< 10^{-6} \text{ kg m}^{-2} \text{ s}^{-1}$ ) it is absolutely essential to change the default value in order to get reasonable results.
- `keep_reference_dataset` (`bool`, default: False): If True, keep the reference dataset in the output. If False, drop the reference dataset.
- `exclude` (`list of str`): Exclude specific datasets from this preprocessor. Note that this option is only available in the recipe, not when using `esmvalcore.preprocessor.bias()` directly (e.g., in another python script). If the reference dataset has been excluded, an error is raised.

Example:

```
preprocessors:
 preproc_bias:
 bias:
 bias_type: relative
 denominator_mask_threshold: 1e-8
 keep_reference_dataset: true
 exclude: [CanESM2]
```

See also `esmvalcore.preprocessor.bias()`.

### distance\_metric

This function calculates a distance metric with respect to a given reference dataset. For this, exactly one input dataset needs to be declared as `reference_for_metric: true` in the recipe, e.g.,

```
datasets:
- {dataset: CanESM5, project: CMIP6, ensemble: r1i1p1f1, grid: gn}
- {dataset: CESM2, project: CMIP6, ensemble: r1i1p1f1, grid: gn}
- {dataset: MIROC6, project: CMIP6, ensemble: r1i1p1f1, grid: gn}
- {dataset: ERA-Interim, project: OBS6, tier: 3, type: reanaly, version: 1,
 reference_for_metric: true}
```

In the example above, ERA-Interim is used as reference dataset for the distance metric calculation.

It is also possible to use the output from the *Multi-model statistics* or *Ensemble statistics* preprocessor as reference dataset. In this case, make sure to use `reference_for_metric: true` for each dataset that will be used to create the reference dataset and use the option `keep_input_datasets: false` for the multi-dataset preprocessor. For example:

```
datasets:
- {dataset: CanESM5, group: ref, reference_for_metric: true}
- {dataset: CESM2, group: ref, reference_for_metric: true}
- {dataset: MIROC6, group: notref}

preprocessors:
 calculate_distance_metric:
 custom_order: true
 multi_model_statistics:
 statistics: [mean]
 span: overlap
 groupby: [group]
 keep_input_datasets: false
 distance_metric:
 metric: emd
```

Here, the EMD metric of MIROC6 is calculated relative to the the multi-model mean from the models CanESM5 and CESM2.

All datasets need to have the same shape and coordinates. To ensure this, the preprocessors `esmvalcore.preprocessor.regrid()` and/or `esmvalcore.preprocessor.regrid_time()` might be helpful.

The `distance_metric` preprocessor supports the following arguments in the recipe:

- `metric (str)`: Distance metric that is calculated. Must be one of
  - `'rmse'`: Unweighted root mean square error.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - r_i)^2}$$

- `'weighted_rmse'`: Weighted root mean square error.

$$WRMSE = \sqrt{\sum_{i=1}^N w_i (x_i - r_i)^2}$$

- `'pearsonr'`: Unweighted Pearson correlation coefficient.

$$r = \frac{\sum_{i=1}^N (x_i - \bar{x})(r_i - \bar{r})}{\sqrt{\sum_{i=1}^N (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^N (r_i - \bar{r})^2}}$$

- 'weighted\_pearsonr': **Weighted Pearson correlation coefficient**.

$$r = \frac{\sum_{i=1}^N w_i (x_i - \bar{x})(r_i - \bar{r})}{\sqrt{\sum_{i=1}^N w_i (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^N w_i (r_i - \bar{r})^2}}$$

- 'emd': **Unweighted Earth mover's distance (EMD)**. The EMD is also known as first Wasserstein metric  $W_1$ , which is a metric that measures distance between two probability distributions. For this, discrete probability distributions of the input data are created through binning, which are then used as input for the Wasserstein metric. The metric is also known as *Earth mover's distance* since, intuitively, it can be seen as the minimum “cost” of turning one pile of earth into another one (pile of earth = probability distribution). This is also known as *optimal transport* problem. Formally, this can be described with a joint probability distribution (or *optimal transport matrix*) (whose marginals are the input distributions) that minimizes the “transportation cost”:

$$W_1 = \min_{\gamma \in \mathbb{R}_+^{n \times n}} \sum_{i,j} \gamma_{ij} |X_i - R_j|$$

with  $\sum_j \gamma_{ij} = p_X(X_i)$ ;  $\sum_i \gamma_{ij} = p_R(R_j)$

- 'weighted\_emd': **Weighted Earth mover's distance**. Similar to the unweighted EMD (see above), but here weights are considered when calculating the probability distributions (i.e., instead of 1, each element provides a weight in the bin count; see also `weights` argument of `numpy.histogram()`).

Here,  $x_i$  and  $r_i$  are samples of a variable of interest and a corresponding reference, respectively (a bar over a variable denotes its arithmetic/weighted mean [the latter for weighted metrics]). Capital letters ( $X_i$  and  $R_i$ ) refer to bin centers of a discrete probability distribution with values  $p_X(X_i)$  or  $p_R(R_i)$  and a number of bins  $n$  (see the argument `n_bins` below) that has been derived for the variables  $x$  and  $r$  through binning. The bins range from the minimum to the maximum value calculated over both the variable of interest and the reference; thus,  $X_i = R_i$  for all  $i$ .  $w_i$  are weights that sum to one (see note below) and  $N$  is the total number of samples.

#### **Note**

Metrics starting with `weighted_` will calculate weighted distance metrics if possible. Currently, the following `coords` (or any combinations that include them) will trigger weighting: `time` (will use lengths of time intervals as weights) and `latitude` (will use cell area weights). Time weights are always calculated from the input data. Area weights can be given as supplementary variables to the recipe (`areacella` or `areacello`, see [Defining supplementary variables \(ancillary variables and cell measures\)](#)) or calculated from the input data (this only works for regular grids). By default, **NO** supplementary variables will be used; they need to be explicitly requested in the recipe.

- `coords` (**list** of **str**, default: `None`): Coordinates over which the distance metric is calculated. If `None`, calculate the metric over all coordinates, which results in a scalar cube.
- `keep_reference_dataset` (**bool**, default: `True`): If `True`, also calculate the distance of the reference dataset with itself. If `False`, drop the reference dataset.
- `exclude` (**list** of **str**): Exclude specific datasets from this preprocessor. Note that this option is only available in the recipe, not when using `esmvalcore.preprocessor.distance_metric()` directly (e.g., in another python script). If the reference dataset has been excluded, an error is raised.
- Other parameters are directly used for the metric calculation. The following keyword arguments are supported:
  - `rmse` and `weighted_rmse`: none.

- *pearsonr* and *weighted\_pearsonr*: `mdtol`, `common_mask` (all keyword arguments are passed to `iris.analysis.stats.pearsonr()`, see that link for more details on these arguments). Note: in contrast to `pearsonr()`, `common_mask=True` by default.
- *emd* and *weighted\_emd*: `n_bins` = number of bins used to create discrete probability distribution of data before calculating the EMD (`int`, default: 100).

Example:

```
preprocessors:
 preproc_pearsonr:
 distance_metric:
 metric: weighted_pearsonr
 coords: [latitude, longitude]
 keep_reference_dataset: true
 exclude: [CanESM2]
 common_mask: true
```

See also `esmvalcore.preprocessor.distance_metric()`.

### 3.2.21 Other

Miscellaneous functions that do not belong to any of the other categories.

#### `align_metadata`

This function sets cube metadata to entries from a specific target project. This is useful to align variable metadata of different projects prior to performing multi-model operations (e.g., *Multi-model statistics*). For example, standard names differ for some variables between CMIP5 and CMIP6 which would prevent the calculation of multi-model statistics between CMIP5 and CMIP6 data.

The `align_metadata` preprocessor supports the following arguments in the recipe:

- `target_project` (`str`): Project from which target metadata is read.
- `target_mip` (`str`; optional): MIP table from which target metadata is read. If not given, use the MIP tables of the corresponding variables defined in the recipe.
- `target_short_name` (`str`; optional): Variable short name from which target metadata is read. If not given, use the short names of the corresponding variables defined in the recipe.
- `target_branding_suffix` (`str`; optional): Variable branding suffix from which target metadata is read. If not given, use the branding suffixes of the corresponding variables defined in the recipe.
- `strict` (`str`; optional, default: `True`): If `True`, raise an error if desired metadata cannot be read for variable `target_short_name` of MIP table `target_mip` and project `target_project`. If `False`, no error is raised.

Example:

```
preprocessors:
 calculate_multi_model_statistics:
 align_metadata:
 target_project: CMIP6
 multi_model_statistics:
 span: overlap
 statistics: [mean, median]
```

See also `esmvalcore.preprocessor.align_metadata()`.

### `cumulative_sum`

This function calculates cumulative sums along a given coordinate.

The `cumulative_sum` preprocessor supports the following arguments in the recipe:

- `coord` (`str`): Coordinate over which the cumulative sum is calculated. Must be 0D or 1D.
- `weights` (array-like, `bool`, or `None`, default: `None`): Weights for the calculation of the cumulative sum. Each element in the data is multiplied by the corresponding weight before summing. Can be an array of the same shape as the input data, `False` or `None` (no weighting), or `True` (calculate the weights from the coordinate bounds; only works if each coordinate point has exactly 2 bounds).
- `method` (`str`, default: `"sequential"`): Method used to perform the cumulative sum. Only relevant if the cube has `lazy data`. See `dask.array.cumsum()` for details.

Example:

```
preprocessors:
 preproc_cumulative_sum:
 cumulative_sum:
 coord: time
 weights: true
```

See also `esmvalcore.preprocessor.cumulative_sum()`.

### `clip`

This function clips data values to a certain minimum, maximum or range. The function takes two arguments:

- `minimum`: Lower bound of range. Default: `None`
- `maximum`: Upper bound of range. Default: `None`

The example below shows how to set all values below zero to zero.

```
preprocessors:
 clip:
 minimum: 0
 maximum: null
```

### `histogram`

This function calculates histograms.

The `histogram` preprocessor supports the following arguments in the recipe:

- `coords` (`list` of `str`, default: `None`): Coordinates over which the histogram is calculated. If `None`, calculate the histogram over all coordinates. The shape of the output cube will be  $(x_1, x_2, \dots, n\_bins)$ , where  $x_i$  are the dimensions of the input cube not appearing in `coords` and  $n\_bins$  is the number of bins.
- `bins` (`int` or sequence of `float`, default: 10): If `bins` is an `int`, it defines the number of equal-width bins in the given `bin_range`. If `bins` is a sequence, it defines a monotonically increasing array of bin edges, including the rightmost edge, allowing for non-uniform bin widths.
- `bin_range` (`tuple` of `float` or `None`, default: `None`): The lower and upper range of the bins. If `None`, `bin_range` is simply `(cube.core_data().min(), cube.core_data().max())`. Values outside the range are ignored. The first element of the range must be less than or equal to the second. `bin_range` affects the automatic bin computation as well if `bins` is an `int` (see description for `bins` above).

- **weights** (array-like, `bool`, or `None`, default: `None`): Weights for the histogram calculation. Each value in the input data only contributes its associated weight towards the bin count (instead of 1). Weights are normalized before entering the calculation if *normalization* is 'integral' or 'sum'. Can be an array of the same shape as the input data, `False` or `None` (no weighting), or `True`. In the latter case, weighting will depend on *coords*, and the following coordinates will trigger weighting: *time* (will use lengths of time intervals as weights) and *latitude* (will use cell area weights). Time weights are always calculated from the input data. Area weights can be given as supplementary variables in the recipe (*areacella* or *areacello*, see *Defining supplementary variables (ancillary variables and cell measures)*) or calculated from the input data (this only works for regular grids). By default, **NO** supplementary variables will be used; they need to be explicitly requested in the recipe.
- **normalization** (`None`, 'sum', or 'integral', default: `None`): If `None`, the result will contain the number of samples in each bin. If 'integral', the result is the value of the probability *density* function at the bin, normalized such that the integral over the range is 1. If 'sum', the result is the value of the probability *mass* function at the bin, normalized such that the sum over the whole range is 1. Normalization will be applied across *coords*, not the entire cube.

Example:

```
preprocessors:
 preproc_histogram:
 histogram:
 coords: [latitude, longitude]
 bins: 12
 bin_range: [100.0, 150.0]
 weights: true
 normalization: sum
```

See also `esmvalcore.preprocessor.histogram()`.

### 3.2.22 Information on maximum memory required

In the most general case, we can set upper limits on the maximum memory the analysis will require:

$M_s = (R + N) \times F_{\text{eff}} - F_{\text{eff}}$  - when no multi-model analysis is performed;

$M_m = (2R + N) \times F_{\text{eff}} - 2F_{\text{eff}}$  - when multi-model analysis is performed;

where

- $M_s$ : maximum memory for non-multimodel module
- $M_m$ : maximum memory for multi-model module
- $R$ : computational efficiency of module;  $R$  is typically 2-3
- $N$ : number of datasets
- $F_{\text{eff}}$ : average size of data per dataset where  $F_{\text{eff}} = e \times f \times F$  where  $e$  is the factor that describes how lazy the data is ( $e = 1$  for fully realized data) and  $f$  describes how much the data was shrunk by the immediately previous module, e.g. time extraction, area selection or level extraction; note that for `fix_data`  $f$  relates only to the time extraction, if data is exact in time (no time selection)  $f = 1$  for `fix_data` so for cases when we deal with a lot of datasets  $R + N \approx N$ , data is fully realized, assuming an average size of 1.5GB for 10 years of 3D netCDF data,  $N$  datasets will require:

$M_s = 1.5 \times (N - 1)$  GB

$M_m = 1.5 \times (N - 2)$  GB

As a rule of thumb, the maximum required memory at a certain time for multi-model analysis could be estimated by multiplying the number of datasets by the average file size of all the datasets; this memory intake is high but also

assumes that all data is fully realized in memory; this aspect will gradually change and the amount of realized data will decrease with the increase of disk use.

## DIAGNOSTIC SCRIPT INTERFACES

In order to communicate with diagnostic scripts, ESMValCore uses YAML files. The YAML files provided by ESMValCore to the diagnostic script tell the diagnostic script the settings that were provided in the recipe and where to find the pre-processed input data. On the other hand, the YAML file provided by the diagnostic script to ESMValCore tells ESMValCore which pre-processed data was used to create what plots. The latter is optional, but needed for recording provenance.

### 4.1 Provenance

When ESMValCore (the `esmvaltool` command) runs a recipe, it will first find all data and run the default preprocessor steps plus any additional preprocessing steps defined in the recipe. Next it will run the diagnostic script defined in the recipe and finally it will store provenance information. Provenance information is stored in the [W3C PROV XML format](#). To read in and extract information, or to plot these files, the `prov` Python package can be used. In addition to provenance information, a caption is also added to the plots.

### 4.2 Information provided by ESMValCore to the diagnostic script

To provide the diagnostic script with the information it needs to run (e.g. location of input data, various settings), the ESMValCore creates a YAML file called `settings.yml` and provides the path to this file as the first command line argument to the diagnostic script.

The most interesting settings provided in this file are

```
run_dir: /path/to/recipe_output/run/diagnostic_name/script_name
work_dir: /path/to/recipe_output/work/diagnostic_name/script_name
plot_dir: /path/to/recipe_output/plots/diagnostic_name/script_name
input_files:
 - /path/to/recipe_output/preproc/diagnostic_name/ta/metadata.yml
 - /path/to/recipe_output/preproc/diagnostic_name/pr/metadata.yml
```

Custom settings in the script section of the recipe will also be made available in this file.

There are three directories defined:

- `run_dir` use this for storing temporary files
- `work_dir` use this for storing NetCDF files containing the data used to make a plot
- `plot_dir` use this for storing plots

Finally `input_files` is a list of YAML files, containing a description of the preprocessed data. Each entry in these YAML files is a path to a preprocessed file in NetCDF format, with a list of various attributes. An example preprocessor metadata.yml file could look like this:

```

? /path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_GFDL-ESM2G_Amon_historical_
↪r1i1p1_T2Ms_pr_2000-2002.nc
: alias: GFDL-ESM2G
 cmor_table: CMIP5
 dataset: GFDL-ESM2G
 diagnostic: diagnostic_name
 end_year: 2002
 ensemble: r1i1p1
 exp: historical
 filename: /path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_GFDL-ESM2G_Amon_
↪historical_r1i1p1_T2Ms_pr_2000-2002.nc
 frequency: mon
 institute: [NOAA-GFDL]
 long_name: Precipitation
 mip: Amon
 modeling_realm: [atmos]
 preprocessor: preprocessor_name
 project: CMIP5
 recipe_dataset_index: 1
 reference_dataset: MPI-ESM-LR
 short_name: pr
 standard_name: precipitation_flux
 start_year: 2000
 units: kg m-2 s-1
 variable_group: pr
? /path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_MPI-ESM-LR_Amon_historical_
↪r1i1p1_T2Ms_pr_2000-2002.nc
: alias: MPI-ESM-LR
 cmor_table: CMIP5
 dataset: MPI-ESM-LR
 diagnostic: diagnostic_name
 end_year: 2002
 ensemble: r1i1p1
 exp: historical
 filename: /path/to/recipe_output/preproc/diagnostic1/pr/CMIP5_MPI-ESM-LR_Amon_
↪historical_r1i1p1_T2Ms_pr_2000-2002.nc
 frequency: mon
 institute: [MPI-M]
 long_name: Precipitation
 mip: Amon
 modeling_realm: [atmos]
 preprocessor: preprocessor_name
 project: CMIP5
 recipe_dataset_index: 2
 reference_dataset: MPI-ESM-LR
 short_name: pr
 standard_name: precipitation_flux
 start_year: 2000
 units: kg m-2 s-1
 variable_group: pr

```

### 4.3 Information provided by the diagnostic script to ESMValCore

After the diagnostic script has finished running, ESMValCore will try to store provenance information. In order to link the produced files to input data, the diagnostic script needs to store a YAML file called `diagnostic_provenance.yml` in its `run_dir`.

For every output file (netCDF files, plot files, etc.) produced by the diagnostic script, there should be an entry in the `diagnostic_provenance.yml` file. The name of each entry should be the path to the file. Each output file entry should at least contain the following items:

- `ancestors` a list of input files used to create the plot.
- `caption` a caption text for the plot.

Each file entry can also contain items from the categories defined in the file `esmvaltool/config_references.yml`. The short entries will automatically be replaced by their longer equivalent in the final provenance records. It is possible to add custom provenance information by adding custom items to entries.

An example `diagnostic_provenance.yml` file could look like this

```
? /path/to/recipe_output/work/diagnostic_name/script_name/CMIP5_GFDL-ESM2G_Amon_
↪historical_r1i1p1_pr_2000-2002_mean.nc
: ancestors: [/path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_GFDL-ESM2G_Amon_
↪historical_r1i1p1_pr_2000-2002.nc]
 authors: [andela_bouwe, righi_mattia]
 caption: Average Precipitation between 2000 and 2002 according to GFDL-ESM2G.
 domains: [global]
 plot_types: [zonal]
 references: [acknow_project]
 statistics: [mean]

? /path/to/recipe_output/plots/diagnostic_name/script_name/CMIP5_GFDL-ESM2G_Amon_
↪historical_r1i1p1_pr_2000-2002_mean.png
: ancestors: [/path/to/recipe_output/preproc/diagnostic_name/pr/CMIP5_GFDL-ESM2G_Amon_
↪historical_r1i1p1_pr_2000-2002.nc]
 authors: [andela_bouwe, righi_mattia]
 caption: Average Precipitation between 2000 and 2002 according to GFDL-ESM2G.
 domains: [global]
 plot_types: ['zonal']
 references: [acknow_project]
 statistics: [mean]
```

You can check whether your diagnostic script successfully provided the provenance information to the ESMValCore by checking the following points:

- for each output file in the `work_dir` and `plot_dir`, a file with the same name, but ending with `_provenance.xml` is created
- the output file is shown on the `index.html` page
- there were no warning messages in the log related to provenance

See [Recording provenance](#) for more extensive usage notes.



## DEVELOPMENT

To get started developing, have a look at our *contribution guidelines*. This chapter describes how to implement the most commonly contributed new features.

### 5.1 Preprocessor function

Preprocessor functions are located in *esmvalcore.preprocessor*. To add a new preprocessor function, start by finding a likely looking file to add your function to in *esmvalcore/preprocessor*. Create a new file in that directory if you cannot find a suitable place.

The function should look like this:

```
def example_preprocessor_function(
 cube,
 example_argument,
 example_optional_argument=5,
):
 """Compute an example quantity.

 A more extensive explanation of the computation can be added here. Add
 references to scientific literature if available.

 Parameters

 cube: iris.cube.Cube
 Input cube.

 example_argument: str
 Example argument, the value of this argument can be provided in the
 recipe. Describe what valid values are here. In this case, a valid
 argument is the name of a dimension of the input cube.

 example_optional_argument: int, optional
 Another example argument, the value of this argument can optionally
 be provided in the recipe. Describe what valid values are here.

 Returns

 iris.cube.Cube
 The result of the example computation.
 """
```

(continues on next page)

(continued from previous page)

```
Replace this with your own computation
cube = cube.collapsed(example_argument, iris.analysis.MEAN)

return cube
```

The above function needs to be imported in the file `esmvalcore/preprocessor/__init__.py`:

```
from ._example_module import example_preprocessor_function

__all__ = [
...
'example_preprocessor_function',
...
]
```

The location in the `__all__` list above determines the default order in which preprocessor functions are applied, so carefully consider where you put it and ask for advice if needed.

The preprocessor function above can then be used from the *Recipe section: preprocessors* like this:

```
preprocessors:
 example_preprocessor:
 example_preprocessor_function:
 example_argument: median
 example_optional_argument: 6
```

The optional argument (in this example: `example_optional_argument`) can be omitted in the recipe.

### 5.1.1 Lazy and real data

Preprocessor functions should support both *real and lazy data*. This is vital for supporting the large datasets that are typically used with the ESMValCore. If the data of the incoming cube has been realized (i.e. `cube.has_lazy_data()` returns `False` so `cube.core_data()` is a `NumPy` array), the returned cube should also have realized data. Conversely, if the incoming cube has lazy data (i.e. `cube.has_lazy_data()` returns `True` so `cube.core_data()` is a `Dask array`), the returned cube should also have lazy data. Note that `NumPy` functions will often call their `Dask` equivalent if it exists and if their input array is a `Dask array`, and vice versa.

Note that preprocessor functions should preferably be small and just call the relevant `iris` code. Code that is more involved, e.g. lots of work with `Numpy` and `Dask` arrays, and more broadly applicable, should be implemented in `iris` instead.

### 5.1.2 Metadata

Preprocessor functions may change the metadata of datasets. An obvious example is `convert_units()`, which changes units. If cube metadata is changed in a preprocessor function, the `metadata.yml` file is automatically updated with this information. The following attributes are taken into account:

| Attribute in metadata.yml file | Updated from                                        |
|--------------------------------|-----------------------------------------------------|
| <code>standard_name</code>     | <code>iris.cube.Cube.standard_name</code>           |
| <code>long_name</code>         | <code>iris.cube.Cube.long_name</code>               |
| <code>short_name</code>        | <code>iris.cube.Cube.var_name</code>                |
| <code>units</code>             | <code>iris.cube.Cube.units</code>                   |
| <code>frequency</code>         | <code>iris.cube.Cube.attributes['frequency']</code> |

If a given cube property is `None`, the corresponding attribute is updated with an empty string (' '). If a cube property is not given, the corresponding attribute is not updated.

### 5.1.3 Documentation

The documentation in the function docstring will be shown in the *Preprocessor functions* chapter. In addition, you should add documentation on how to use the new preprocessor function from the recipe in [doc/recipe/preprocessor.rst](#) so it is shown in the *Preprocessor* chapter. See the introduction to *Documentation* for more information on how to best write documentation.

### 5.1.4 Tests

Tests should be implemented for new or modified preprocessor functions. For an introduction to the topic, see *Tests*.

#### Unit tests

To add a unit test for the preprocessor function from the example above, create a file called `tests/unit/preprocessor/_example_module/test_example_preprocessor_function.py` and add the following content:

```

"""Test function `esmvalcore.preprocessor.example_preprocessor_function`."""
import cf_units
import dask.array as da
import iris
import numpy as np
import pytest

from esmvalcore.preprocessor import example_preprocessor_function

@pytest.mark.parametrize('lazy', [True, False])
def test_example_preprocessor_function(lazy):
 """Test that the computed result is as expected."""

 # Construct the input cube
 data = np.array([1, 2], dtype=np.float32)
 if lazy:
 data = da.asarray(data, chunks=(1,))
 cube = iris.cube.Cube(
 data,
 var_name='tas',
 units='K',
)
 cube.add_dim_coord(
 iris.coords.DimCoord(
 np.array([0.5, 1.5], dtype=np.float64),
 bounds=np.array([[0, 1], [1, 2]], dtype=np.float64),
 standard_name='time',
 units=cf_units.Unit('days since 1950-01-01 00:00:00',
 calendar='gregorian'),
),
 0,
)

 # Compute the result

```

(continues on next page)

(continued from previous page)

```

result = example_preprocessor_function(cube, example_argument='time')

Check that lazy data is returned if and only if the input is lazy
assert result.has_lazy_data() is lazy

Construct the expected result cube
expected = iris.cube.Cube(
 np.array(1.5, dtype=np.float32),
 var_name='tas',
 units='K',
)
expected.add_aux_coord(
 iris.coords.AuxCoord(
 np.array([1], dtype=np.float64),
 bounds=np.array([[0, 2]], dtype=np.float64),
 standard_name='time',
 units=cf_units.Unit('days since 1950-01-01 00:00:00',
 calendar='gregorian'),
))
expected.add_cell_method(
 iris.coords.CellMethod(method='mean', coords=('time',)))

Compare the result of the computation with the expected result
print('result:', result)
print('expected result:', expected)
assert result == expected

```

In this test we used the decorator `pytest.mark.parametrize` to test two scenarios, with both lazy and realized data, with a single test.

### Sample data tests

The idea of adding *sample data tests* is to check that preprocessor functions work with realistic data. This also provides an easy way to add regression tests, though these should preferably be implemented as unit tests instead, because using the sample data for this purpose is slow. To add a test using the sample data, create a file `tests/sample_data/preprocessor/example_preprocessor_function/test_example_preprocessor_function.py` and add the following content:

```

"""Test function `esmvalcore.preprocessor.example_preprocessor_function`."""
from pathlib import Path

import iris
import pytest

from esmvalcore.preprocessor import example_preprocessor_function

esmvaltool_sample_data = pytest.importorskip("esmvaltool_sample_data")

@pytest.mark.use_sample_data
def test_example_preprocessor_function():
 """Regression test to check that the computed result is as expected."""

```

(continues on next page)

(continued from previous page)

```

Load an example input cube
cube = esmvaltool_sample_data.load_timeseries_cubes(mip_table='Amon')[0]

Compute the result
result = example_preprocessor_function(cube, example_argument='time')

filename = Path(__file__).with_name('example_preprocessor_function.nc')
if not filename.exists():
 # Create the file the expected result if it doesn't exist
 iris.save(result, target=str(filename))
 raise FileNotFoundError(
 f'Reference data was missing, wrote new copy to {filename}')

Load the expected result cube
expected = iris.load_cube(str(filename))

Compare the result of the computation with the expected result
print('result:', result)
print('expected result:', expected)
assert result == expected

```

This will use a file from the sample data repository as input. The first time you run the test, the computed result will be stored in the file `tests/sample_data/preprocessor/example_preprocessor_function/example_preprocessor_function.nc`. Any subsequent runs will re-load the data from file and check that it did not change. Make sure the stored results are small, i.e. smaller than 100 kilobytes, to keep the size of the ESMValCore repository small.

### 5.1.5 Using multiple datasets as input

The name of the first argument of the preprocessor function should in almost all cases be `cube`. Only when implementing a preprocessor function that uses all datasets as input, the name of the first argument should be `products`. If you would like to implement this type of preprocessor function, start by having a look at the existing functions, e.g. `esmvalcore.preprocessor.multi_model_statistics()` or `esmvalcore.preprocessor.mask_fillvalues()`.

## 5.2 Fixing data

The baseline case for ESMValCore input data is fully *CMOR compliant* data that is read using Iris' `iris.load_raw()`. ESMValCore also allows for some departures from compliance (see *Customizing checker strictness*). Beyond that situation, some datasets (either model or observations) contain (known) errors that would normally prevent them from being processed. The issues can be in the metadata describing the dataset and/or in the actual data. Typical examples of such errors are missing or wrong attributes (e.g. attribute "units" says 1e-9 but data are actually in 1e-6), missing or mislabeled coordinates (e.g. "lev" instead of "plev" or missing coordinate bounds like "lat\_bnds") or problems with the actual data (e.g. cloud liquid water only instead of sum of liquid + ice as specified by the CMIP data request). As an extreme case, some data sources simply are not NetCDF files and must go through some other data load function. ESMValCore can apply on-the-fly fixes to such datasets when issues can be fixed automatically.

In addition, some datasets are supported in their native (i.e., non CMOR-compliant) format through fixes. This is implemented for a set of *Datasets in native format*. A detailed description of how to include new native datasets is given *below*.

The following sections provide details on how to design such fixes.

**Note**

**CMORizer scripts.** Support for many observational and reanalysis datasets is also possible through a priori reformatting by [CMORizer scripts](#) in the ESMValTool, which are rather relevant for datasets of small volume

### 5.2.1 Fix structure

Fixes are Python classes stored in `esmvalcore/cmor/_fixes/[PROJECT]/[DATASET].py` that derive from `esmvalcore.cmor._fixes.fix.Fix` and are named after the short name of the variable they fix. You can also use the names of mip tables (e.g., Amon, Lmon, Omon, etc.) if you want the fix to be applied to all variables of that table in the dataset or AllVars if you want the fix to be applied to the whole dataset.

**Warning**

Be careful to replace any `-` with `_` in your dataset name. We need this replacement to have proper python module names.

The fixes are automatically loaded and applied when the dataset is preprocessed. They are a special type of *pre-processor function*, called by the preprocessor functions `esmvalcore.preprocessor.fix_file()`, `esmvalcore.preprocessor.fix_metadata()`, and `esmvalcore.preprocessor.fix_data()`.

The Fix class provides the following attributes:

- `Fix.vardef`: *VariableInfo* object that corresponds to the variable fixed by the fix.
- `Fix.extra_facets`: `dict` that contains all facets of the corresponding dataset fixed by the fix (see `esmvalcore.dataset.Dataset.facets`).
- `Fix.session`: *Session* object that includes configuration and directory information.

### 5.2.2 Fixing a dataset

To illustrate the process of creating a fix we are going to construct a new one from scratch for a fictional dataset. We need to fix a CMIPX model called PERFECT-MODEL that is reporting a missing latitude coordinate for variable tas.

#### Check the output

Next to the error message, you should see some info about the iris cube: size, coordinates. In our example it looks like this:

```
air_temperature/ (K) (time: 312; altitude: 90; longitude: 180)
 Dimension coordinates:
 time x - -
 altitude - x -
 longitude - - x
 Auxiliary coordinates:
 day_of_month x - -
 day_of_year x - -
 month_number x - -
 year x - -
 Attributes:
 {'cmor_table': 'CMIPX', 'mip': 'Amon', 'short_name': 'tas', 'frequency': 'mon'}}
```

So now the mistake is clear: the latitude coordinate is badly named and the fix should just rename it.

## Create the fix

We start by creating the module file. In our example the path will be `esmvalcore/cmor/_fixes/CMIPX/PERFECT_MODEL.py`. If it already exists just add the class to the file, there is no limit in the number of fixes we can have in any given file.

Then we have to create the class for the fix deriving from `esmvalcore.cmor._fixes.Fix`

```

"""Fixes for PERFECT-MODEL."""
from esmvalcore.cmor.fix import Fix

class tas(Fix):
 """Fixes for tas variable."""

```

Next we must choose the method to use between the ones offered by the Fix class:

- `fix_file`: you need to fix errors that prevent loading the data with Iris or perform operations that are more efficient with other packages (e.g., loading files with lots of variables is much faster with Xarray than Iris).
- `fix_metadata`: you want to change something in the cube that is not the data (e.g., variable or coordinate names, data units).
- `fix_data`: you need to fix the data. Beware: coordinates data values are part of the metadata.

In our case we need to rename the coordinate altitude to latitude, so we will implement the `fix_metadata` method:

```

"""Fixes for PERFECT-MODEL."""
from esmvalcore.cmor.fix import Fix

class tas(Fix):
 """Fixes for tas variable."""

 def fix_metadata(self, cubes):
 """
 Fix metadata for tas.

 Fix the name of the latitude coordinate, which is called altitude
 in the original file.
 """

 # Sometimes Iris will interpret the data as multiple cubes.
 # Good CMOR datasets will only show one but we support the
 # multiple cubes case to be able to fix the errors that are
 # leading to that extra cubes.
 # In our case this means that we can safely assume that the
 # tas cube is the first one
 tas_cube = cubes[0]
 latitude = tas_cube.coord('altitude')

 # Fix the names. Latitude values, units and
 latitude.short_name = 'lat'
 latitude.standard_name = 'latitude'
 latitude.long_name = 'latitude'
 return cubes

```

This will fix the error. The next time you run ESMValTool you will find that the error is fixed on the fly and, hopefully, your recipe will run free of errors. The `cubes` argument to the `fix_metadata` method will contain all cubes loaded

from a single input file. Some care may need to be taken that the right cube is selected and fixed in case multiple cubes are created. Usually this happens when a coordinate is mistakenly loaded as a cube, because the input data does not follow the [CF Conventions](#).

Sometimes other errors can appear after you fix the first one because they were hidden by it. In our case, the latitude coordinate could have bad units or values outside the valid range for example. Just extend your fix to address those errors.

## Finishing

Chances are that you are not the only one that wants to use that dataset and variable. Other users could take advantage of your fixes as soon as possible. Please, create a separated pull request for the fix and submit it.

It will also be very helpful if you just scan a couple of other variables from the same dataset and check if they share this error. In case that you find that it is a general one, you can change the fix name to the corresponding mip table name (e.g., `Amon`, `Lmon`, `Omon`, etc.) so it gets executed for all variables in that table in the dataset or to `AllVars` so it gets executed for all variables in the dataset. If you find that this is shared only by a handful of similar vars you can just make the fix for those new vars derive from the one you just created:

```
"""Fixes for PERFECT-MODEL."""
from esmvalcore.cmor.fix import Fix

class tas(Fix):
 """Fixes for tas variable."""

 def fix_metadata(self, cubes):
 """
 Fix metadata for tas.

 Fix the name of the latitude coordinate, which is called altitude
 in the original file.
 """
 # Sometimes Iris will interpret the data as multiple cubes.
 # Good CMOR datasets will only show one but we support the
 # multiple cubes case to be able to fix the errors that are
 # leading to that extra cubes.
 # In our case this means that we can safely assume that the
 # tas cube is the first one
 tas_cube = cubes[0]
 latitude = tas_cube.coord('altitude')

 # Fix the names. Latitude values, units and
 latitude.short_name = 'lat'
 latitude.standard_name = 'latitude'
 latitude.long_name = 'latitude'
 return cubes

class ps(tas):
 """Fixes for ps variable."""
```

### 5.2.3 Common errors

The above example covers one of the most common cases: variables / coordinates that have names that do not match the expected. But there are some others that use to appear frequently. This section describes the most common cases.

#### Bad units declared

It is quite common that a variable declares to be using some units but the data is stored in another. This can be solved by overwriting the units attribute with the actual data units.

```
def fix_metadata(self, cubes):
 cube = self.get_cube_from_list(cubes)
 cube.units = 'real_units'
```

Detecting this error can be tricky if the units are similar enough. It also has a good chance of going undetected until you notice strange results in your diagnostic.

For the above example, it can be useful to access the variable definition and associated coordinate definitions as provided by the CMOR table. For example:

```
def fix_metadata(self, cubes):
 cube = self.get_cube_from_list(cubes)
 cube.units = self.vardef.units
```

To learn more about what is available in these definitions, see: [esmvalcore.cmor.table.VariableInfo](#) and [esmvalcore.cmor.table.CoordinateInfo](#).

#### Coordinates missing

Another common error is to have missing coordinates. Usually it just means that the file does not follow the CF-conventions and Iris can therefore not interpret it.

If this is the case, you should see a warning from the ESMValTool about discarding some cubes in the fix metadata step. Just before that warning you should see the full list of cubes as read by Iris. If that list contains your missing coordinate you can create a fix for this model:

```
def fix_metadata(self, cubes):
 coord_cube = cubes.extract_cube('COORDINATE_NAME')
 # Usually this will correspond to an auxiliary coordinate
 # because the most common error is to forget adding it to the
 # coordinates attribute
 coord = iris.coords.AuxCoord(
 coord_cube.data,
 var_name=coord_cube.var_name,
 standard_name=coord_cube.standard_name,
 long_name=coord_cube.long_name,
 units=coord_cube.units,
)

 # It may also have bounds as another cube
 coord.bounds = cubes.extract_cube('BOUNDS_NAME').data

 data_cube = cubes.extract_cube('VAR_NAME')
 data_cube.add_aux_coord(coord, DIMENSIONS_INDEX_TUPLE)
 return [data_cube]
```

## 5.2.4 Customizing checker strictness

The baseline case for ESMValCore input data is fully *CMOR compliant* data and this is checked when data is loaded by `esmvalcore.dataset.Dataset.load()`.

However, it is possible to disable these checks completely by *configuring the project* so it uses the `esmvalcore.cmor.table.NoInfo` CMOR table, or to adjust the strictness of the checks using the *configuration option* `check_level`.

The data checker classifies its issues using four different levels of severity. From highest to lowest:

- **CRITICAL**: issues that most of the time will have severe consequences.
- **ERROR**: issues that usually lead to unexpected errors, but can be safely ignored sometimes.
- **WARNING**: something is not up to the standard but is unlikely to have consequences later.
- **DEBUG**: any info that the checker wants to communicate. Regardless of checker strictness, those will always be reported as debug messages.

Users can have control about which levels of issues are interpreted as errors, and therefore make the checker fail or warnings or debug messages. For this purpose there is an optional *configuration option* `check_level` that can take a number of values, listed below from the lowest level of strictness to the highest:

- **ignore**: all issues, regardless of severity, will be reported as warnings. Checker will never fail. Use this at your own risk.
- **relaxed**: only CRITICAL issues are treated as errors. We recommend not to rely on this mode, although it can be useful if there are errors preventing the run that you are sure you can manage on the diagnostics or that will not affect you.
- **default**: fail if there are any CRITICAL or ERROR issues (DEFAULT); Provides a good measure of safety.
- **strict**: fail if there are any warnings, this is the highest level of strictness. Mostly useful for checking datasets that you have produced, to be sure that future users will not be distracted by inoffensive warnings.

### Warning

While it is possible to work with datasets that are not described in a CMOR table or only partially follow the CMOR standards, the *preprocessor functions* and *diagnostics* have been designed to work with CMORized data and may not work as expected with non-CMORized data.

Our ambition is that *preprocessor functions* support data that follows the *CF Conventions*.

## 5.2.5 Add support for new native datasets

This section describes how to add support for additional native datasets. You can choose to host this new data source either under a dedicated project or under project `native6`.

### CMOR Table Configuration

An example of a CMOR table configuration for projects used for native datasets is given here:

Listing 1: Example native data format projects in `defaults/cmor_tables.yml`

```
projects:
 # Observational and reanalysis data that can be read in its native format by
 ↪ ESMValCore.
 native6:
```

(continues on next page)

(continued from previous page)

```

cmor_table:
 type: esmvalcore.cmor.table.CMIP6Info
 paths:
 - cmip6/Tables
 - cmip6-custom
 strict: false
 # Data from selected climate models that can be read in its native format by
 ↪ ESMValCore.
ACCESS:
 cmor_table:
 type: esmvalcore.cmor.table.CMIP6Info
 paths:
 - cmip6/Tables
 - cmip6-custom
 strict: false

```

The option `strict: false` is convenient for these projects if you want to make use of the feature that looks in all tables instead of only the one specified by the `mip` facet in the *recipe* or *Dataset*: like this, a custom variable only needs to be defined for a single `mip` and can then be used with all `mip` s.

## Locate data

To allow ESMValCore to locate the data files, use the following steps:

- If you want to use the `native6` project, recommended for datasets whose input files can be easily moved to the usual `native6` directory structure given by

Listing 2: `native6` standard directory organization in `data-local-esmvaltool.yml`

```

Read native6, OBS6, and OBS data from the filesystem on a personal computer.
projects:
 # Data that can be read in its native format by ESMValCore.
 native6:
 data:
 local:
 type: esmvalcore.io.local.LocalDataSource
 rootpath: ~/climate_data
 dirname_template: "Tier{tier}/{dataset}/{version}/{frequency}/{short_name}"
 filename_template: "*.nc"

```

this is preferred.

- If you want to use a dedicated project for your native dataset (this is usually the case for native model output):  
A new entry for the project needs to be added under *Project-specific configuration*. For example, for the ICON model, create a new project `ICON` and define its data sources:

Listing 3: `ICON` standard directory organization in `data-native-icon.yml`

```

Read data from the ICON model in its native format.
projects:
 ICON:
 data:

```

(continues on next page)

(continued from previous page)

```

icon: &icon
 type: esmvalcore.io.local.LocalDataSource
 rootpath: ~/climate_data
 dirname_template: "{exp}"
 filename_template: "{exp}_{var_type}*.nc"
 ignore_warnings:
 - message: "Failed to create 'height' dimension coordinate: The 'height'
↳DimCoord bounds array must be strictly monotonic."
 module: iris
 icon-outdata:
 <<: *icon
 dirname_template: "{exp}/outdata"
 icon-output:
 <<: *icon
 dirname_template: "{exp}/output"

```

and a CMOR table configuration:

Listing 4: ICON CMOR table configuration from defaults/  
cmor\_tables.yml

```

projects:
 ICON:
 cmor_table:
 type: esmvalcore.cmor.table.CMIP6Info
 paths:
 - cmip6/Tables
 - cmip6-custom
 strict: false

```

To find your ICON data that is, for example, located in files like `~/climate_data/amip/amip_atm_2d_ml_20000101T000000Z.nc`, use the following dataset entry in your recipe:

```

datasets:
 - {project: ICON, dataset: ICON, exp: amip}

```

Please note the duplication of the name `ICON` in `project` and `dataset`, which is necessary to comply with ESMValTool's data finding and CMORizing functionalities. For other native models, `dataset` could also refer to a subversion of the model. Note that it is possible to predefine facets via *extra facets*. In this `ICON` example, the facet `var_type` is predefined for many variables.

## Fix native data

To ensure that the native dataset has the correct metadata and data (i.e., that it is CMOR-compliant), use *dataset fixes*. This is where the actual CMORization takes place. For example, a native6 dataset fix for ERA5 is located [here](#), and the `ICON` fix is located [here](#).

ESMValTool also provides a base class `NativeDatasetFix` that provides convenient functions useful for all native dataset fixes. An example for its usage can be found [here](#).

## Extra facets for native datasets

If necessary, so-called *Extra Facets* can be provided which allow to cope e.g. with variable naming issues for finding files or additional information that is required for the fixes. See *Use of extra facets in fixes* for more details on this. An example of such a configuration file for IPSL-CM6 is given [here](#).

### 5.2.6 Use of extra facets in fixes

Extra facets are a mechanism to provide additional information for certain kinds of data. The general approach is described in *Extra Facets*. Here, we describe how they can be used in fixes to mold data into the form required by the applicable standard. For example, if the input data is part of an observational product that delivers surface temperature with a variable name of `t2m` inside a file named `2m_temperature_1950_monthly.nc`, but the same variable is called `tas` in the applicable standard, a fix can be created that reads the original variable from the correct file, and provides a renamed variable to the rest of the processing chain.

Normally, the applicable standard for variables is CMIP6.

For more details, refer to existing uses of this feature as examples, as e.g. *for IPSL-CM6*.

## 5.3 Deriving a variable

The variable derivation preprocessor module allows to derive variables which are not in the CMIP standard data request using standard variables as input. This is a special type of *preprocessor function*. All derivation scripts are located in `esmvalcore/preprocessor/_derive/`. A typical example looks like this:

```

"""Derivation of variable `dummy`."""

from iris.cube import Cube, CubeList

from esmvalcore.typing import Facets, FacetValue

from ._baseclass import DerivedVariableBase

class DerivedVariable(DerivedVariableBase):
 """Derivation of variable `dummy`."""

 @staticmethod
 def required(project: str) -> list[Facets]:
 """Declare the variables needed for derivation."""
 mip = "fx"
 if project == "CMIP6":
 mip = "Ofx"
 required = [
 {"short_name": "var_a"},
 {"short_name": "var_b", "mip": mip, "optional": True},
]
 return required

 @staticmethod
 def calculate(cubes: CubeList) -> Cube:
 """Compute `dummy`."""

 # `cubes` is a CubeList containing all required variables.

```

(continues on next page)

(continued from previous page)

```

cube = do_something_with(cubes)

Return single cube at the end
return cube

```

The static function `required(project)` returns a *list* of *Facets* containing all required variables for deriving the derived variable. Its only argument is the `project` of the specific dataset. In this particular example script, the derived variable `dummy` is derived from `var_a` and `var_b`. It is possible to specify arbitrary attributes for each required variable, e.g. `var_b` uses the `mip_fx` (or `Ofx` in the case of CMIP6) instead of the original one of `dummy`. Note that you can also declare a required variable as `optional=True`, which allows the skipping of this particular variable during data extraction. For example, this is useful for `fx` variables which are often not available for observational datasets. Otherwise, the tool will fail if not all required variables are available for all datasets.

The actual derivation takes place in the static function `calculate(cubes)` which returns a single *Cube* containing the derived variable. Its only argument `cubes` is a *CubeList* containing all required variables.

If no MIP table entry for the derived variable exists for the given `mip`, the tool will also look in other `mip` tables for the same `project` to find the definition of derived variables. To contribute a completely new derived variable, it is necessary to define a name for it and to provide the corresponding CMOR table. This is to guarantee the proper metadata definition is attached to the derived data. Such custom CMOR tables are collected as part of the ESMValCore package. An example entry for CMIP6 looks like:

Listing 5: The `asr` variable definition in the custom CMIP6 table at `esmvalcore/cmor/tables/cmip6/tables/cmip6-custom/CMIP6_custom.json`.

```

"asr": {
 "cell_measures": "area: areacella",
 "cell_methods": "time: mean",
 "dimensions": "longitude latitude time",
 "long_name": "Absorbed shortwave radiation",
 "modeling_realm": "atmos",
 "out_name": "asr",
 "positive": "down",
 "standard_name": "",
 "type": "real",
 "units": "W m-2"
},

```

More details on this can also be found in the documentation of the *CMOR table configuration*.

## CONTRIBUTIONS ARE VERY WELCOME

We greatly value contributions of any kind. Contributions could include, but are not limited to documentation improvements, bug reports, new or improved code, scientific and technical code reviews, infrastructure improvements, mailing list and chat participation, community help/building, education and outreach. We value the time you invest in contributing and strive to make the process as easy as possible. If you have suggestions for improving the process of contributing, please do not hesitate to propose them.

If you have a bug or other issue to report or just need help, please open an issue on the [issues tab on the ESMValCore github repository](#).

If you would like to contribute a new *preprocessor function*, *derived variable*, *fix for a dataset*, or another new feature, please discuss your idea with the development team before getting started, to avoid double work and/or disappointment later. A good way to do this is to open an [issue](#) on GitHub.

### 6.1 Getting started

See [Installation from source](#) for instructions on how to set up a development installation.

New development should preferably be done in the [ESMValCore](#) GitHub repository. The default git branch is `main`. Use this branch to create a new feature branch from and make a pull request against. This [page](#) offers a good introduction to git branches, but it was written for BitBucket while we use GitHub, so replace the word BitBucket by GitHub whenever you read it.

It is recommended that you open a [draft pull request](#) early, as this will cause [CircleCI to run the unit tests](#), [pre-commit.ci to analyse your code](#), and [readthedocs to build the documentation](#). It's also easier to get help from other developers if your code is visible in a pull request.

[Make small pull requests](#), the ideal pull requests changes just a few files and adds/changes no more than 100 lines of production code. The amount of test code added can be more extensive, but changes to existing test code should be made sparingly.

#### 6.1.1 Design considerations

When making changes, try to respect the current structure of the program. If you need to make major changes to the structure of program to add a feature, chances are that you have either not come up with the most optimal design or the feature is not a very good fit for the tool. Discuss your feature with the [@ESMValGroup/esmvaltool-coreteam](#) in an [issue](#) to find a solution.

Please keep the following considerations in mind when programming:

- Changes should preferably be *backward compatible*.
- Apply changes gradually and change no more than a few files in a single pull request, but do make sure every pull request in itself brings a meaningful improvement. This reduces the risk of breaking existing functionality and making *backward incompatible* changes, because it helps you as well as the reviewers of your pull request to better understand what exactly is being changed.

- *Preprocessor functions* are Python functions (and not classes) so they are easy to understand and implement for scientific contributors.
- No additional CMOR checks should be implemented inside preprocessor functions, though it is our ambition to make preprocessor functions work with any data that follows the [CF Conventions](#). The input cube is fixed and confirmed to follow the specification in [esmvalcore/cmor/tables](#) before applying any other preprocessor functions. This design helps to keep the preprocessor functions and diagnostics scripts that use the preprocessed data from the tool simple and reliable. See [CMOR table configuration](#) for the mapping from project in the recipe to the relevant CMOR table.
- The ESMValCore package is based on [iris](#). Preprocessor functions should preferably be small and just call the relevant iris code. This automatically ensures that the preprocessor functions work with data that follows the CF Conventions. Code that is more involved and more broadly applicable than just in the ESMValCore, should be implemented in iris instead.
- Any settings in the recipe that can be checked before loading the data should be checked at the [task creation stage](#). This avoids that users run a recipe for several hours before finding out they made a mistake in the recipe. No data should be processed or files written while creating the tasks.
- CMOR checks should provide a good balance between reliability of the tool and ease of use. Several [levels of strictness of the checks](#) are available to facilitate this.
- Keep your code short and simple: we would like to make contributing as easy as possible. For example, avoid implementing complicated class inheritance structures and [boilerplate](#) code.
- If you find yourself copy-pasting a piece of code and making minor changes to every copy, instead put the repeated bit of code in a function that you can reuse, and provide the changed bits as function arguments.
- Be careful when changing existing unit tests to make your new feature work. You might be breaking existing features if you have to change existing tests.

Finally, if you would like to improve the design of the tool, discuss your plans with the [@ESMValGroup/esmvaltool-coreteam](#) to make sure you understand the current functionality and you all agree on the new design.

## 6.2 Checklist for pull requests

To clearly communicate up front what is expected from a pull request, we have the following checklist. Please try to do everything on the list before requesting a review. If you are unsure about something on the list, please ask the [@ESMValGroup/tech-reviewers](#) or [@ESMValGroup/science-reviewers](#) for help by commenting on your (draft) pull request or by starting a new [discussion](#).

In the ESMValTool community we use [pull request reviews](#) to ensure all code and documentation contributions are of good quality. The icons indicate whether the item will be checked during the [Technical review](#) or [Scientific review](#).

- The new functionality is *relevant and scientifically sound*
- *The pull request has a descriptive title and labels*
- Code is written according to the [code quality guidelines](#)
- and [Documentation](#) is available
- Unit *tests* have been added
- Changes are *backward compatible*
- Changed *dependencies have been added or removed correctly*
- The *list of authors* is up to date
- The *checks shown below the pull request* are successful

Pull requests introducing a change that causes a recipe to no longer run successfully (*breaking change*), or which results in scientifically significant changes in results (*science change*) require additional items to be reviewed defined in the [backward compatibility policy](#). These include in particular:

- Instructions for the release notes to assist *recipe developers* to adapt their recipe in light of the *backward-incompatible change* available.
- General instructions for *recipe developers* working on *user recipes* to enable them to adapt their code related to *backward-incompatible changes* available (see [ESMValTool\\_Tutorial: issue #263](#)).
- Core development team tagged to notify them of the *backward-incompatible change*, and give at least 2 weeks for objections to be raised before merging to the main branch. If a strong objection is raised the backward-incompatible change should not be merged until the objection is resolved.
- Information required for the “*backward-incompatible changes*” section in the PR that introduces the *backward-incompatible change* available.

## 6.3 Scientific relevance

The proposed changes should be relevant for the larger scientific community. The implementation of new features should be scientifically sound; e.g. the formulas used in new preprocessor functions should be accompanied by the relevant references and checked for correctness by the scientific reviewer. The [CF Conventions](#) as well as additional standards imposed by [CMIP](#) should be followed whenever possible.

## 6.4 Pull request title and label

The title of a pull request should clearly describe what the pull request changes. If you need more text to describe what the pull request does, please add it in the description. Add [one or more labels](#) to your pull request to indicate the type of change. At least one of the following [labels](#) should be used: *bug*, *deprecated feature*, *fix for dataset*, *preprocessor*, *cmor*, *api*, *testing*, *documentation* or *enhancement*.

The titles and labels of pull requests are used to compile the [Changelog](#), therefore it is important that they are easy to understand for people who are not familiar with the code or people in the project. Descriptive pull request titles also makes it easier to find back what was changed when, which is useful in case a bug was introduced.

## 6.5 Code quality

To increase the readability and maintainability of the ESMValCore source code, we aim to adhere to best practices and coding standards.

We include checks for Python and yaml files, most of which are described in more detail in the sections below. This includes checks for invalid syntax and formatting errors. [Pre-commit](#) is a handy tool that can run all of these checks automatically just before you commit your code. It knows which tool to run for each filetype, and therefore provides a convenient way to check your code. Install the pre-commit hooks by running

```
pre-commit install
```

to make sure your code is formatted correctly and does not contain mistakes whenever you commit some changes.

### 6.5.1 Python

The standard document on best practices for Python code is [PEP8](#) and there is [PEP257](#) for code documentation. We make use of [numpy style docstrings](#) to document Python functions that are part of the [ESMValCore API Reference](#).

To check if your code adheres to the standard, go to the directory where the repository is cloned, e.g. `cd ESMValCore`, and run `pre-commit`:

```
pre-commit run --all
```

We use `ruff` to automatically format the code and to check for certain bugs and `mypy` for checking that `type hints` are correct. Note that `type hints` are optional, but if you do choose to add them, they should be correct. Both `ruff` and `mypy` are automatically run by `pre-commit`.

When you make a pull request, adherence to the Python development best practices is checked by `pre-commit.ci`.

## 6.5.2 YAML

We use `yamllint` to check that YAML files do not contain mistakes. This is automatically run by `pre-commit`.

## 6.5.3 Any text file

A generic tool to check for common spelling mistakes is `codespell`. This is automatically run by `pre-commit`.

# 6.6 Documentation

The documentation lives on `docs.esmvaltool.org`.

## 6.6.1 Adding documentation

The documentation is built by `readthedocs` using `Sphinx`. There are two main ways of adding documentation:

1. As written text in the directory `doc`. When writing `reStructuredText` (`.rst`) files, please try to limit the line length to 80 characters and always start a sentence on a new line. This makes it easier to review changes to documentation on GitHub.
2. As docstrings or comments in code. For Python code, only the `docstrings` of Python modules, classes, and functions that are mentioned in `doc/api` are used to generate the online documentation. This results in the *ES-MValCore API Reference*. The standard document with best practices on writing docstrings is `PEP257`. For the API documentation, we make use of `numpy` style `docstrings`.

## 6.6.2 What should be documented

Functionality that is visible to users should be documented. Any documentation that is visible on `readthedocs` should be well written and adhere to the standards for documentation. Examples of this include:

- The *recipe*
- Preprocessor *functions* and their *use from the recipe*
- *Configuration options*
- *Installation*
- *Output files*
- *Command line interface*
- *Diagnostic script interfaces*
- *The experimental Python interface*

Note that:

- For functions that compute scientific results, comments with references to papers and/or other resources as well as formula numbers should be included.
- When making changes to/introducing a new preprocessor function, also update the *preprocessor documentation*.

- There is no need to write complete numpy style documentation for functions that are not visible in the [ESMValCore API Reference](#) chapter on readthedocs. However, adding a docstring describing what a function does is always a good idea. For short functions, a one-line docstring is usually sufficient, but more complex functions might require slightly more extensive documentation.

When reviewing a pull request, always check that documentation is easy to understand and available in all expected places.

### 6.6.3 How to build and view the documentation

Whenever you make a pull request or push new commits to an existing pull request, [readthedocs](#) will automatically build the documentation. The link to the documentation will be shown in the list of checks below your pull request. Click 'Details' behind the check `docs/readthedocs.org:esmvalcore` to preview the documentation. If all checks were successful, you may need to click 'Show all checks' to see the individual checks.

To build the documentation on your own computer, go to the directory where the repository was cloned and run

```
sphinx-build doc doc/build
```

or

```
sphinx-build -Ea doc doc/build
```

to build it from scratch.

Alternatively, the `pixi` task `doc` can be used:

```
pixi run doc
```

Make sure that your newly added documentation builds without warnings or errors and looks correctly formatted. [readthedocs](#) will build the documentation with the command:

```
sphinx-build -W doc doc/build
```

to catch mistakes that can be detected automatically.

The configuration file for `Sphinx` is `doc/conf.py` and the configuration file for ReadTheDocs is `.readthedocs.yaml`.

See [Integration with the ESMValCore documentation](#) for information on how the ESMValCore documentation is integrated into the complete ESMValTool project documentation on readthedocs.

When reviewing a pull request, always check that the documentation checks shown below the pull request were successful.

## 6.7 Tests

To check that the code works correctly, there tests available in the `tests` directory. We use `pytest` to write and run our tests.

Contributions to ESMValCore should be covered by unit tests. Have a look at the existing tests in the `tests` directory for inspiration on how to write your own tests. If you do not know how to start with writing unit tests, ask the [@ESMValGroup/tech-reviewers](#) for help by commenting on the pull request and they will try to help you. It is also recommended that you have a look at the `pytest` documentation at some point when you start writing your own tests.

### 6.7.1 Running tests

To run the tests on your own computer, go to the directory where the repository is cloned and run the command

```
pytest
```

or

```
pixi run test
```

If you are working in a place with a slow internet connection, you may want to skip tests that require an internet connection by adding `-m 'not online'` to the command. To only run tests from a single file, run the command

```
pytest tests/unit/test_some_file.py
```

If you would like to avoid loading the default pytest configuration from `pyproject.toml` because this can be a bit slow for running just a few tests, use

```
pytest -c /dev/null tests/unit/test_some_file.py
```

Use

```
pytest --help
```

for more information on the available commands.

Whenever you make a pull request or push new commits to an existing pull request, the tests in the `tests` directory of the branch associated with the pull request will be run automatically on [CircleCI](#). The results appear at the bottom of the pull request. Click on 'Details' for more information on a specific test job.

When reviewing a pull request, always check that all test jobs on [CircleCI](#) were successful.

### 6.7.2 Test coverage

To check which parts of your code are [covered by unit tests](#), run the command

```
pytest --cov
```

and open the file `test-reports/coverage_html/index.html` and browse to the relevant file. Note that tracking code coverage slows down the test runs, therefore it is disabled by default and needs to be requested by providing `pytest` with the `--cov` flag.

[CircleCI](#) will upload the coverage results from running the tests to [Codecov](#). [codecov](#) is a service that will comment on pull requests with a summary of the test coverage. If [codecov](#) reports that the coverage has decreased, check the report and add additional tests. Alternatively, it is also possible to view code coverage on [CircleCI](#) (open the `tests` job and click the ARTIFACTS tab). To see some of the results on [CircleCI](#) or [Codecov](#), you may need to log in; you can do so using your [GitHub](#) account.

When reviewing a pull request, always check that new code is covered by unit tests and [codecov](#) reports an increased coverage.

### 6.7.3 Sample data

New or modified preprocessor functions should preferably also be tested using the sample data. These tests are located in `tests/sample_data`. Please mark new tests that use the sample data with the decorator `@pytest.mark.use_sample_data`.

The `ESMValTool_sample_data` repository contains samples of CMIP6 data for testing ESMValCore. The `ESMValTool-sample-data` package is installed as part of the developer dependencies. The size of the package is relatively small (~100 MB), so it can be easily downloaded and distributed.

Preprocessing the sample data can be time-consuming, so some intermediate results are cached by pytest to make the tests run faster. If you suspect the tests are failing because the cache is invalid, clear it by running

```
pytest --cache-clear
```

To avoid running the time consuming tests that use sample data altogether, run

```
pytest -m "not use_sample_data"
```

## 6.7.4 Automated testing

Whenever you make a pull request or push new commits to an existing pull request, the tests in the `tests` of the branch associated with the pull request will be run automatically on [CircleCI](#).

Every night, more extensive tests are run to make sure that problems with the installation of the tool are discovered by the development team before users encounter them. These nightly tests have been designed to follow the installation procedures described in the documentation, e.g. in the [Installation](#) chapter. The nightly tests are run using both CircleCI and GitHub Actions. The result of the tests ran by CircleCI can be seen on the [CircleCI project page](#) and the result of the tests ran by GitHub Actions can be viewed on the [Actions tab](#) of the repository (to learn more about the Github-hosted runners, please have a look the [documentation](#)).

The configuration of the tests run by CircleCI can be found in the directory `.circleci`, while the configuration of the tests run by GitHub Actions can be found in the directory `.github/workflows`.

## 6.8 Backward compatibility

The ESMValCore package is used by many people to run their recipes. Many of these recipes are maintained in the public [ESMValTool](#) repository, but there are also users who choose not to share their work there. While our commitment is first and foremost to users who do share their recipes in the ESMValTool repository, we still try to be nice to all of the ESMValCore users.

### Note

The [backward compatibility policy](#) outlines the key principles on backward compatibility and additional guidance on handling *backward-incompatible changes*. This policy applies to both, ESMValCore and ESMValTool.

When making changes, e.g. to the *recipe format*, the *diagnostic script interface*, the public *Python API*, or the *configuration format*, keep in mind that this may affect many users. To keep the tool user friendly, try to avoid making changes that are not backward compatible, i.e. changes that require users to change their existing recipes, diagnostics, configuration files, or scripts.

If you really must change the public interfaces of the tool, always discuss this with the [@ESMValGroup/esmvaltool-coreteam](#). Try to deprecate the feature first by issuing an [ESMValCoreDeprecationWarning](#) using the `warnings` module and schedule it for removal two [minor versions](#) from the upcoming release. For example, when you deprecate a feature in a pull request that will be included in version 2.5, that feature should be removed in version 2.7:

```
import warnings

from esmvalcore.exceptions import ESMValCoreDeprecationWarning
```

(continues on next page)

(continued from previous page)

```

Other code

def func(x, deprecated_option=None):
 """Deprecate deprecated_option."""
 if deprecated_option is not None:
 deprecation_msg = (
 "The option ``deprecated_option`` has been deprecated in "
 "ESMValCore version 2.5 and is scheduled for removal in "
 "version 2.7. Add additional text (e.g., description of "
 "alternatives) here.")
 warnings.warn(deprecation_msg, ESMValCoreDeprecationWarning, stacklevel=2)

Other code

```

Mention the version in which the feature will be removed in the deprecation message. Label the pull request with the `deprecated feature` label. When deprecating a feature, please follow up by actually removing the feature in due course.

If you must make backward incompatible changes, you need to update the available recipes in ESMValTool and link the ESMValTool pull request(s) in the ESMValCore pull request description. You can ask the [@ESMValGroup/esmvaltool-recipe-maintainers](#) for help with updating existing recipes, but please be considerate of their time. You should tag the [@ESMValGroup/esmvaltool-coreteam](#) to notify them of the backward-incompatible change, and give at least 2 weeks for objections to be raised before merging to the main branch. If a strong objection is raised the backwards-incompatible change should not be merged until the objection is resolved.

When reviewing a pull request, always check for backward incompatible changes and make sure they are needed and have been discussed with the [@ESMValGroup/esmvaltool-coreteam](#). Also, make sure the author of the pull request has created the accompanying pull request(s) to update the ESMValTool, before merging the ESMValCore pull request.

## 6.9 Dependencies

We follow the guidelines set out in [SPEC-0](#) regarding the supported Python versions and dependencies.

Before considering adding a new dependency, carefully check that the [license](#) of the dependency you want to add and any of its dependencies are [compatible](#) with the [Apache 2.0](#) license that applies to the ESMValCore. Note that GPL version 2 license is considered incompatible with the Apache 2.0 license, while the compatibility of GPL version 3 license with the Apache 2.0 license is questionable. See this [statement](#) by the authors of the Apache 2.0 license for more information.

When adding or removing dependencies, please consider applying the changes in the following locations in `pyproject.toml`:

- `dependencies` contains dependencies that can be installed from [PyPI](#)
- `[tool.pixi.dependencies]` contains dependencies that can be installed from [conda-forge](#)

it is strongly preferred that those two lists are kept in sync, apart from differences in how packages are named. Run the command `pixi lock` after making changes to the dependencies to update the `pixi.lock` file, which is used to make sure that the same versions of packages are installed for all ESMValCore developers.

There are also three feature groups for development dependencies:

- `[tool.pixi.feature.dev]` contains tools that are useful for development
- `[tool.pixi.feature.doc]` contains tools that are needed to build the documentation
- `[tool.pixi.feature.test]` contains tools that are needed to run the tests

 **Tip**

When reviewing a pull request where dependencies are added or removed, always check that the changes have been applied to both the PyPI and the conda-forge dependencies and that the `pixi.lock` file has been updated by running `pixi lock` after the changes were made.

## 6.10 List of authors

If you make a contribution to ESMValCore and you would like to be listed as an author (e.g. on [Zenodo](#)), please add your name to the list of authors in `CITATION.cff` and generate the entry for the `.zenodo.json` file by running the commands

```
pip install cffconvert
cffconvert --ignore-suspect-keys --outputformat zenodo --outfile .zenodo.json
```

Presently, this method unfortunately discards entries *communities* and *grants* from that file; please restore them manually, or alternately proceed with the addition manually

## 6.11 Pull request checks

To check that a pull request is up to standard, several automatic checks are run when you make a pull request. Read more about it in the [Tests](#) and [Documentation](#) sections. Successful checks have a green ✓ in front, a means the check failed.

If you need help with the checks, please ask the technical reviewer of your pull request for help. Ask [@ESMValGroup/tech-reviewers](#) if you do not have a technical reviewer yet.

If the checks are broken because of something unrelated to the current pull request, please check if there is an open issue that reports the problem. Create one if there is no issue yet. You can attract the attention of the [@ESMValGroup/esmvaltool-coreteam](#) by mentioning them in the issue if it looks like no-one is working on solving the problem yet. The issue needs to be fixed in a separate pull request first. After that has been merged into the `main` branch and all checks on this branch are green again, merge it into your own branch to get the tests to pass.

When reviewing a pull request, always make sure that all checks were successful. See the section on [code\\_quality](#) for more information. Never merge a pull request with failing pre-commit, CircleCI, or readthedocs checks.



## HOW-TO GUIDES

**Note**

This is work in progress as we are currently working on adding more how-to guides to this section. For a more detailed overview of what how-to guides should cover, you can head to the [Diataxis](#) page.

### 7.1 How to make a release

The release manager makes the release, assisted by the release manager of the previous release, or if that person is not available, another previous release manager. Perform the steps listed below with two persons, to reduce the risk of error.

**Note**

The previous release manager ensures the current release manager has the required administrative permissions to make the release. Consider the following services: [conda-forge](#), [DockerHub](#), [PyPI](#), and [readthedocs](#).

The release of ESMValCore is tied to the release of ESMValTool. The detailed steps can be found in the ESMValTool [documentation](#). To start the procedure, ESMValCore gets released as a release candidate to test the recipes in ESMValTool. If bugs are found during the testing phase of the release candidate, make as many release candidates for ESMValCore as needed in order to fix them.

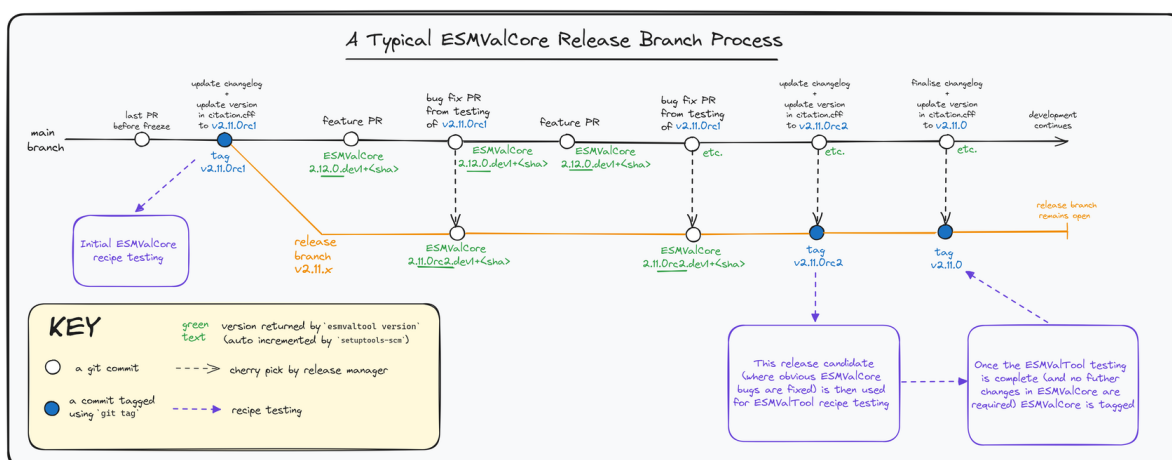


Fig. 1: Example of a release timeline for ESMValCore (in this case for 2.11.0)

To make a new release of the package, be it a release candidate or the final release, follow these steps:

### 7.1.1 1. Check that all tests and builds work

- Check that the [nightly test run on CircleCI](#) was successful.
- Check that the [GitHub Actions test runs](#) were successful.
- Check that the documentation builds successfully on [readthedocs](#).
- Check that the [Docker images](#) are building successfully.

All tests should pass before making a release (branch).

### 7.1.2 2. Create a release branch

Create a branch off the `main` branch and push it to GitHub. Ask someone with administrative permissions to set up branch protection rules for it so only you and the person helping you with the release can push to it.

### 7.1.3 3. Increase the version number

The version number is automatically generated from the information provided by `git` using `[setuptools-scm](https://pypi.org/project/setuptools-scm/)`, but a static version number is stored in `CITATION.cff`. Make sure to update the version number and release date in `CITATION.cff`. See <https://semver.org> for more information on choosing a version number. Make a pull request and get it merged into `main` and cherry pick it into the release branch.

### 7.1.4 4. Add release notes

Use the script `esmvaltool/utis/draft_release_notes.py` to create a draft of the release notes. This script uses the titles and labels of merged pull requests since the previous release. Open a discussion to allow members of the development team to nominate pull requests as highlights. Add the most voted pull requests as highlights at the beginning of changelog. After the highlights section, list any backward incompatible changes that the release may include. The [backward compatibility policy](#) lists the information that should be provided by the developer of any backward incompatible change. Make sure to also list any deprecations that the release may include, as well as a brief description on how to upgrade a deprecated feature. Review the results, and if anything needs changing, change it on GitHub and re-run the script until the changelog looks acceptable. Copy the result to the file `doc/changelog.rst`. Make a pull request and get it merged into `main` and cherry pick it into the release branch.

### 7.1.5 5. Make the (pre-)release on GitHub

Do a final check that all tests on CircleCI and GitHub Actions completed successfully. Then click the [releases](#) tab and create the new release from the release branch (i.e. not from `main`).

Create a tag and tick the *This is a pre-release* box if working with a release candidate.

### 7.1.6 6. Mark the release in the main branch

When the (pre-)release is tagged, it is time to merge the release branch back into `main`. We do this for two reasons, namely, one, to mark the point up to which commits in `main` have been considered for inclusion into the present release, and, two, to inform `setuptools-scm` about the version number so that it creates the correct version number in `main`. However, unlike in a normal merge, we do not want to integrate any of the changes from the release branch into `main`. This is because all changes that should be in both branches, i.e. bug fixes, originate from `main` anyway and the only other changes in the release branch relate to the release itself. To take this into account, we perform the merge in this case on the command line using the [ours merge strategy](#) (`git merge -s ours`), not to be confused with the `ours` option to the `ort` merge strategy (`git merge -X ours`). For details about merge strategies, see the above-linked page. To execute the merge use following sequence of steps

```
git fetch
git checkout main
git pull
git merge -s ours v2.1.x
git push
```

Note that the release branch remains intact and you should continue any work on the release on that branch.

### 7.1.7 7. Create and upload the PyPI package

The package is automatically uploaded to the PyPI by a GitHub action. Note that for security reasons, the upload needs to be approved by someone from the [ESMValGroup/technical-lead-development-team](#) who is not the author of the release.

If the automatic build and upload has failed for some reason, do it manually by following these instructions:

- Check out the tag corresponding to the release, e.g. `git checkout tags/v2.1.0`
- Make sure your current working directory is clean by checking the output of `git status` and by running `git clean -xdf` to remove any files ignored by git.
- Install the required packages: `python3 -m pip install --upgrade build twine`
- Build the package: `python3 -m build` This command should generate two files in the `dist` directory, e.g. `ESMValCore-2.3.1-py3-none-any.whl` and `ESMValCore-2.3.1.tar.gz`.
- Upload the package: `python3 -m twine upload dist/*` You will be prompted for an API token if you have not set this up before, see [here](#) for more information.

You can read more about this in [Packaging Python Projects](#).

### 7.1.8 8. Create the Conda package

The `esmvalcore` package is published on the [conda-forge conda channel](#). This is done via a pull request on the [esmvalcore-feedstock repository](#).

To publish a release candidate, you have to open a pull request yourself. An example for this can be found [here](#). Make sure to use the `rc` branch as the target branch for your pull request and follow all instructions given by the linter bot. The testing of ESMValTool will be performed with the published release candidate.

For the final release, this pull request is automatically opened by a bot. An example pull request can be found [here](#). Follow the instructions by the bot to finalize the pull request. This step mostly contains updating dependencies that have been changed during the last release cycle. Once approved by the [feedstock maintainers](#) they will merge the pull request, which will in turn publish the package on conda-forge some time later. Contact the feedstock maintainers if you want to become a maintainer yourself.

### 7.1.9 9. Check the Docker images

There are two main Docker container images available for ESMValCore on [Dockerhub](#):

- `esmvalgroup/esmvalcore:stable`, built from [docker/Dockerfile](#), this is a tag that is always the same as the latest released version. This image is only built by Dockerhub when a new release is created.
- `esmvalgroup/esmvalcore:development`, built from [docker/Dockerfile.dev](#), this is a tag that always contains the latest conda environment for ESMValCore, including any test dependencies. It is used by [CircleCI](#) to run the unit tests. This speeds up running the tests, as it avoids the need to build the conda environment for every test run. This image is built by Dockerhub every time there is a new commit to the `main` branch on Github.

In addition to the two images mentioned above, there is an image available for every release (e.g. `esmvalgroup/esmvalcore:v2.5.0`). When working on the Docker images, always try to follow the [best practices](#).

After making the release, check that the Docker image for that release has been built correctly by

1. checking that the version tag is available on [Dockerhub](#) and the `stable` tag has been updated,
2. running some recipes with the `stable` tag Docker container, for example one recipe for Python, NCL, and R,
3. running a recipe with a Singularity container built from the `stable` tag.

If there is a problem with the automatically built container image, you can fix the problem and build a new image locally. For example, to [build](#) and [upload](#) the container image for v2.5.0 of the tool run:

```
git checkout v2.5.0
git clean -x
docker build -t esmvalgroup/esmvalcore:v2.5.0 . -f docker/Dockerfile
docker push esmvalgroup/esmvalcore:v2.5.0
```

(when making updates, you may want to add `.post0`, `.post1`, .. to the version number to avoid overwriting an older tag) and if it is the latest release that you are updating, also run

```
docker tag esmvalgroup/esmvalcore:v2.5.0 esmvalgroup/esmvalcore:stable
docker push esmvalgroup/esmvalcore:stable
```

Note that the `docker push` command will overwrite the existing tags on Dockerhub, but the previous container image will remain available as an untagged image.

## REFERENCE GUIDES

**Note**

This is work in progress as we are currently working on adding more reference guides to this section. For a more detailed overview of what reference guides should cover, you can head to the [Diataxis page](#).

## 8.1 Variables and CMOR Tables

ESMValCore has been designed to facilitate working with [Earth System Model](#) data, also known as climate model data. To make it easy to compare and combine data from different climate models, reanalysis datasets, and observational datasets, ESMValCore uses the standardized variables from the [CMOR tables](#) provided by the projects it supports. [CMOR](#) (Climate Model Output Rewriter) is a tool commonly used by climate modelling centers to format their model output according to community standards. The CMOR tables define the standardized variable names, units, coordinates, and other metadata for various climate variables and are typically compiled from a Data Request and a Controlled Vocabulary, e.g. the [CMIP7 CMOR tables](#) are based on the [CMIP7 Data Request](#), and the [CMIP7 Controlled Vocabulary](#). ESMValCore comes bundled with several CMOR tables, which are stored in the directory `esmvalcore/cmor/tables`. It is possible to *configure which CMOR tables are used by ESMValCore*.

The *facets* project, `mip`, `short_name`, and optionally `branding_suffix`, uniquely determine the variable to use. These facets are used to look up the variable in the CMOR table for the project. Compliance with the variable definition from the CMOR table is checked when data is loaded, to avoid unexpected results or errors during data processing. The strictness of these checks can be *configured*. For example, the facets `project: CMIP6`, `mip: Amon`, `short_name: tas` define the near-surface air temperature variable in the CMIP6 Amon table:

Listing 1: The `tas` variable definition in the CMIP6 Amon table at `esmvalcore/cmor/tables/cmip6/Tables/CMIP6_Amon.json`.

```
"tas": {
 "frequency": "mon",
 "modeling_realm": "atmos",
 "standard_name": "air_temperature",
 "units": "K",
 "cell_methods": "area: time: mean",
 "cell_measures": "area: areacella",
 "long_name": "Near-Surface Air Temperature",
 "comment": "near-surface (usually, 2 meter) air temperature",
 "dimensions": "longitude latitude time height2m",
 "out_name": "tas",
 "type": "real",
 "positive": ""
```

(continues on next page)

(continued from previous page)

```

 "valid_min": "",
 "valid_max": "",
 "ok_min_mean_abs": "",
 "ok_max_mean_abs": ""
 },

```

In some cases the `short_name` (called `out_name` in the CMOR tables) of a variable may differ from the name used as a key in the CMOR table. This is always the case for CMIP7, where the `branded variable name` is used, which is composed of the `short_name` followed by an underscore and the `branding_suffix`. For example, the facets `project: CMIP7, mip: atmos, short_name: tas, branding_suffix: tavg-h2m-hxy-u` select one of the near-surface air temperature variables in the CMIP7 `atmos` table:

Listing 2: One of the `tas` variable definitions in the CMIP7 `atmos` table at `esmvalcore/cmor/tables/cmip7/tables/CMIP7_atmos.json`.

```

"tas_tavg-h2m-hxy-u": {
 "cell_measures": "",
 "cell_methods": "area: time: mean",
 "comment": "",
 "dimensions": [
 "longitude",
 "latitude",
 "time",
 "height2m"
],
 "long_name": "Near-Surface Air Temperature",
 "modeling_realm": "atmos",
 "out_name": "tas",
 "positive": "",
 "standard_name": "air_temperature",
 "units": "K"
},

```

For other projects, the facet `branding_suffix` can also be used to distinguish between variables from the same CMOR table that share the same `short_name`, but differ in other aspects, even though these projects do not use branded variables. For example, the `ch4Clim` entry in the CMIP6 `Amon` table can be selected in the recipe by specifying `project: CMIP6, mip: Amon, short_name: ch4, branding_suffix: Clim`:

Listing 3: One of the `ch4` variable definitions in the CMIP6 `Amon` table at `esmvalcore/cmor/tables/cmip6/Tables/CMIP6_Amon.json`.

```

"ch4Clim": {
 "frequency": "monC",
 "modeling_realm": "atmos atmosChem",
 "standard_name": "mole_fraction_of_methane_in_air",
 "units": "mol mol-1",
 "cell_methods": "area: mean time: mean within years time: mean over years",
 "cell_measures": "area: areacella",
 "long_name": "Mole Fraction of CH4",
 "comment": "Mole fraction is used in the construction mole_fraction_of_X_in_
↪Y, where X is a material constituent of Y.",
 "dimensions": "longitude latitude plev19 time2",
 "out_name": "ch4",

```

(continues on next page)

(continued from previous page)

```

 "type": "real",
 "positive": "",
 "valid_min": "",
 "valid_max": "",
 "ok_min_mean_abs": "",
 "ok_max_mean_abs": ""
 },

```

## 8.2 Facets

A facet is a key-value pair that describes a certain property of a dataset and enables [faceted search](#), for example as provided by [ESGF](#). The facets used on ESGF are closely related to the global attributes defined by the [controlled vocabulary](#) used by the various “project”s hosted on ESGF. A “project” is a collection of datasets that share certain properties, e.g. [CMIP7](#) is a project. Each project has its own set of facets that are relevant for that project. The documents linked below provide an overview of the official facets for various projects. They also provide a reference directory structure and file naming convention based on facets, which is used to organise data on local filesystems.

ESMValCore uses “facets” to search for and define input data, both in the [recipe](#) and in the [Dataset](#) object. This allows specifying data without relying on e.g. file names or directory structures, which may vary between computers. ESMValCore uses its own set of facets, which is consistent across all projects it supports.

Here is a mapping from the facet names used in ESMValCore to the corresponding project specific facet names used on ESGF.

### 8.2.1 CMIP7

Official CMIP7 facets.

#### Note

This mapping is preliminary as no CMIP7 data has been published on ESGF yet.

| ESMValCore facet | ESGF facet        |
|------------------|-------------------|
| activity         | activity_id       |
| branding_suffix  | branding_suffix   |
| dataset          | source_id         |
| ensemble         | variant_label     |
| exp              | experiment_id     |
| frequency        | frequency         |
| grid             | grid_label        |
| institute        | institution_id    |
| realm            | realm             |
| region           | region            |
| project          | project / mip_era |
| short_name       | variable_id       |
| version          | version           |

## 8.2.2 CMIP6

Official CMIP6 facets.

| ESMValCore facet | ESGF facet        |
|------------------|-------------------|
| activity         | activity_id       |
| dataset          | source_id         |
| ensemble         | member_id         |
| exp              | experiment_id     |
| frequency        | frequency         |
| grid             | grid_label        |
| institute        | institution_id    |
| mip              | table_id          |
| realm            | realm             |
| project          | project / mip_era |
| short_name       | variable_id       |

## 8.2.3 CMIP5

Official CMIP5 facets. Note that there appear to be differences between the official facets and those used on ESGF. Below we present the facets used on ESGF.

| ESMValCore facet | ESGF facet     |
|------------------|----------------|
| dataset          | model          |
| ensemble         | ensemble       |
| exp              | experiment     |
| frequency        | time_frequency |
| institute        | institute      |
| mip              | cmor_table     |
| realm            | realm          |
| product          | product        |
| project          | project        |
| short_name       | variable       |

## 8.2.4 CMIP3

| ESMValCore facet | ESGF facet     |
|------------------|----------------|
| dataset          | model          |
| ensemble         | ensemble       |
| exp              | experiment     |
| frequency        | time_frequency |
| short_name       | variable       |

## 8.2.5 CORDEX

Official CORDEX-CMIP5 facets. Note that there appear to be differences between the official facets and those used on ESGF. Below we present the facets used on ESGF.

| ESMValCore facet | ESGF facet     |
|------------------|----------------|
| dataset          | rcm_name       |
| driver           | driving_model  |
| domain           | domain         |
| ensemble         | ensemble       |
| exp              | experiment     |
| frequency        | time_frequency |
| institute        | institute      |
| short_name       | variable       |

### 8.2.6 obs4MIPs

Official [obs4MIPs facets](#). Note that obs4MIPs first followed the CMIP5 conventions before switching to the CMIP6 conventions. That means that both conventions are in use depending on when a particular dataset was published. See the CMIP5 and CMIP6 tables above for the mappings.



## ESMVALCORE API REFERENCE

ESMValCore is mostly used as a commandline tool. However, it is also possible to use (parts of) ESMValTool as a library. This section documents the public API of ESMValCore.

### 9.1 CMOR functions

CMOR module.

#### 9.1.1 Checking compliance

Module for checking iris cubes against their CMOR definitions.

##### Classes:

|                                                     |                                                      |
|-----------------------------------------------------|------------------------------------------------------|
| <i>CheckLevels</i> (*values)                        | Level of strictness of the checks.                   |
| <i>CMORCheck</i> (cube, var_info[, frequency, ...]) | Class used to check the CMOR-compliance of the data. |

##### Exceptions:

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <i>CMORCheckError</i> | Exception raised when a cube does not pass the CMORCheck. |
|-----------------------|-----------------------------------------------------------|

##### Functions:

|                                                          |                                                           |
|----------------------------------------------------------|-----------------------------------------------------------|
| <i>cmor_check_metadata</i> (cube, cmor_table, mip, ...)  | Check if metadata conforms to variable's CMOR definition. |
| <i>cmor_check_data</i> (cube, cmor_table, mip, ...)      | Check if data conforms to variable's CMOR definition.     |
| <i>cmor_check</i> (cube, cmor_table, mip, short_name, *) | Check if cube conforms to variable's CMOR definition.     |

**class** `esmvalcore.cmor.check.CheckLevels`(\*values)

Bases: `IntEnum`

Level of strictness of the checks.

##### Attributes:

|              |                                                                 |
|--------------|-----------------------------------------------------------------|
| <i>DEBUG</i> | Report any debug message that the checker wants to communicate. |
|--------------|-----------------------------------------------------------------|

continues on next page

Table 4 – continued from previous page

|                |                                                                    |
|----------------|--------------------------------------------------------------------|
| <i>STRICT</i>  | Fail if there are warnings regarding compliance of CMOR standards. |
| <i>DEFAULT</i> | Fail if cubes present any discrepancy with CMOR standards.         |
| <i>RELAXED</i> | Fail if cubes present severe discrepancies with CMOR standards.    |
| <i>IGNORE</i>  | Do not fail for any discrepancy with CMOR standards.               |

**DEBUG = 1**

Report any debug message that the checker wants to communicate.

**STRICT = 2**

Fail if there are warnings regarding compliance of CMOR standards.

**DEFAULT = 3**

Fail if cubes present any discrepancy with CMOR standards.

**RELAXED = 4**

Fail if cubes present severe discrepancies with CMOR standards.

**IGNORE = 5**

Do not fail for any discrepancy with CMOR standards.

**exception** `esmvalcore.cmor.check.CMORCheckError`

Bases: `Exception`

Exception raised when a cube does not pass the CMORCheck.

**class** `esmvalcore.cmor.check.CMORCheck`(*cube*, *var\_info*, *frequency=None*, *fail\_on\_error=False*, *check\_level=CheckLevels.DEFAULT*)

Bases: `object`

Class used to check the CMOR-compliance of the data.

**Parameters**

- **cube** (*iris.cube.Cube*) – Iris cube to check.
- **var\_info** (*variables\_info.VariableInfo*) – Variable info to check.
- **frequency** (*str*) – Expected frequency for the data. If not given, use the one from the variable information.
- **fail\_on\_error** (*bool*) – If true, CMORCheck stops on the first error. If false, it collects all possible errors before stopping.
- **check\_level** (*CheckLevels*) – Level of strictness of the checks.

**frequency**

Expected frequency for the data.

**Type**

`str`

**Methods:**

|                                                   |                                                     |
|---------------------------------------------------|-----------------------------------------------------|
| <code>check_metadata([logger])</code>             | Check the cube metadata.                            |
| <code>check_data([logger])</code>                 | Check the cube data.                                |
| <code>report_errors()</code>                      | Report detected errors.                             |
| <code>report_warnings()</code>                    | Report detected warnings to the given logger.       |
| <code>report_debug_messages()</code>              | Report detected debug messages to the given logger. |
| <code>has_errors()</code>                         | Check if there are reported errors.                 |
| <code>has_warnings()</code>                       | Check if there are reported warnings.               |
| <code>has_debug_messages()</code>                 | Check if there are reported debug messages.         |
| <code>report(level, message, *args)</code>        | Report a message from the checker.                  |
| <code>report_critical(message, *args)</code>      | Report an error.                                    |
| <code>report_error(message, *args)</code>         | Report a normal error.                              |
| <code>report_warning(message, *args)</code>       | Report a warning level error.                       |
| <code>report_debug_message(message, *args)</code> | Report a debug message.                             |

**check\_metadata**(*logger*: *logging.Logger* | *None* = *None*) → Cube

Check the cube metadata.

It will also report some warnings in case of minor errors.

**Parameters**

**logger** (*logging.Logger* | *None*) – Given logger.

**Returns**

Checked cube.

**Return type**

*iris.cube.Cube*

**Raises**

**CMORCheckError** – If errors are found. If *fail\_on\_error* attribute is set to True, raises as soon as an error is detected. If set to False, it perform all checks and then raises.

**check\_data**(*logger*: *logging.Logger* | *None* = *None*) → Cube

Check the cube data.

Assumes that metadata is correct, so you must call `check_metadata` prior to this.

It will also report some warnings in case of minor errors.

**Parameters**

**logger** (*logging.Logger* | *None*) – Given logger.

**Returns**

Checked cube.

**Return type**

*iris.cube.Cube*

**Raises**

**CMORCheckError** – If errors are found. If *fail\_on\_error* attribute is set to True, raises as soon as an error is detected. If set to False, it perform all checks and then raises.

**report\_errors**()

Report detected errors.

**Raises**

**CMORCheckError** – If any errors were reported before calling this method.

**report\_warnings()**

Report detected warnings to the given logger.

**report\_debug\_messages()**

Report detected debug messages to the given logger.

**has\_errors()**

Check if there are reported errors.

**Returns**

True if there are pending errors, False otherwise.

**Return type**

bool

**has\_warnings()**

Check if there are reported warnings.

**Returns**

True if there are pending warnings, False otherwise.

**Return type**

bool

**has\_debug\_messages()**

Check if there are reported debug messages.

**Returns**

True if there are pending debug messages, False otherwise.

**Return type**

bool

**report(*level*, *message*, \**args*)**

Report a message from the checker.

**Parameters**

- **level** (*CheckLevels*) – Message level
- **message** (*str*) – Message to report
- **args** – String format args for the message

**Raises**

**CMORCheckError** – If fail on error is set, it is thrown when registering an error message

**report\_critical(*message*, \**args*)**

Report an error.

If fail\_on\_error is set to True, raises automatically. If fail\_on\_error is set to False, stores it for later reports.

**Parameters**

- **message** (*str*: *unicode*) – Message for the error.
- **\*args** – arguments to format the message string.

**report\_error(*message*, \**args*)**

Report a normal error.

**Parameters**

- **message** (*str*: *unicode*) – Message for the error.

- **\*args** – arguments to format the message string.

**report\_warning**(*message*, *\*args*)

Report a warning level error.

**Parameters**

- **message** (*str*: *unicode*) – Message for the warning.
- **\*args** – arguments to format the message string.

**report\_debug\_message**(*message*, *\*args*)

Report a debug message.

**Parameters**

- **message** (*str*: *unicode*) – Message for the debug logger.
- **\*args** – arguments to format the message string

`esmvalcore.cmor.check.cmor_check_metadata`(*cube*: *Cube*, *cmor\_table*: *str*, *mip*: *str*, *short\_name*: *str*, \*  
(*Keyword-only parameters separator (PEP 3102)*),  
*branding\_suffix*: *str* | *None* = *None*, *frequency*: *str* | *None* = *None*,  
*check\_level*: *CheckLevels* = *CheckLevels.DEFAULT*)  
→ *Cube*

Check if metadata conforms to variable's CMOR definition.

None of the checks at this step will force the cube to load the data.

**Parameters**

- **cube** (*Cube*) – Data cube to check.
- **cmor\_table** (*str*) – CMOR definitions to use (i.e., the variable's project).
- **mip** (*str*) – Variable's MIP.
- **short\_name** (*str*) – Variable's short name.
- **branding\_suffix** (*str* | *None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table. Used by the CMIP7 project.
- **frequency** (*str* | *None*) – Data frequency. If not given, use the one from the CMOR table of the variable.
- **check\_level** (*CheckLevels*) – Level of strictness of the checks.

**Returns**

Checked cube.

**Return type**

`iris.cube.Cube`

`esmvalcore.cmor.check.cmor_check_data`(*cube*: *Cube*, *cmor\_table*: *str*, *mip*: *str*, *short\_name*: *str*, \*  
*branding\_suffix*: *str* | *None* = *None*, *frequency*: *str* | *None* = *None*,  
*check\_level*: *CheckLevels* = *CheckLevels.DEFAULT*) → *Cube*

Check if data conforms to variable's CMOR definition.

**Parameters**

- **cube** (*Cube*) – Data cube to check.
- **cmor\_table** (*str*) – CMOR definitions to use (i.e., the variable's project).
- **mip** (*str*) – Variable's MIP.

- **short\_name** (*str*) – Variable's short name
- **branding\_suffix** (*str* | *None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table. Used by the CMIP7 project.
- **frequency** (*str* | *None*) – Data frequency. If not given, use the one from the CMOR table of the variable.
- **check\_level** (`CheckLevels`) – Level of strictness of the checks.

**Returns**

Checked cube.

**Return type**

`iris.cube.Cube`

`esmvalcore.cmor.check.cmor_check(cube: Cube, cmor_table: str, mip: str, short_name: str, *, branding_suffix: str | None = None, frequency: str | None = None, check_level: CheckLevels = CheckLevels.DEFAULT) → Cube`

Check if cube conforms to variable's CMOR definition.

Equivalent to calling `cmor_check_metadata()` and `cmor_check_data()` consecutively.

**Parameters**

- **cube** (*Cube*) – Data cube to check.
- **cmor\_table** (*str*) – CMOR definitions to use (i.e., the variable's project).
- **mip** (*str*) – Variable's MIP.
- **short\_name** (*str*) – Variable's short name.
- **branding\_suffix** (*str* | *None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table. Used by the CMIP7 project.
- **frequency** (*str* | *None*) – Data frequency. If not given, use the one from the CMOR table of the variable.
- **check\_level** (`CheckLevels`) – Level of strictness of the checks.

**Returns**

Checked cube.

**Return type**

`iris.cube.Cube`

### 9.1.2 Automatically fixing issues

Apply automatic fixes for known errors in cmorized data.

All functions in this module will work even if no fixes are available for the given dataset. Therefore is recommended to apply them to all variables to be sure that all known errors are fixed.

**Functions:**

|                                                                |                                                              |
|----------------------------------------------------------------|--------------------------------------------------------------|
| <code>fix_data(cube, short_name, project, dataset, mip)</code> | Fix cube data if fixes are required.                         |
| <code>fix_file(file, short_name, project, dataset, ...)</code> | Fix files before loading them into a <code>CubeList</code> . |
| <code>fix_metadata(cubes, short_name, project, ...)</code>     | Fix cube metadata if fixes are required.                     |

`esmvalcore.cmor.fix.fix_data(cube: Cube, short_name: str, project: str, dataset: str, mip: str, frequency: str | None = None, session: Session | None = None, **extra_facets: Any) → Cube`

Fix cube data if fixes are required.

This method assumes that metadata is already fixed and checked.

This method collects all the relevant fixes (including generic ones) for a given variable and applies them.

#### Parameters

- **cube** (*Cube*) – Cube to fix.
- **short\_name** (*str*) – Variable's short name.
- **project** (*str*) – Project of the dataset.
- **dataset** (*str*) – Name of the dataset.
- **mip** (*str*) – Variable's MIP.
- **frequency** (*str* | *None*) – Variable's data frequency, if available.
- **session** (*Session* | *None*) – Current session which includes configuration and directory information.
- **\*\*extra\_facets** (*Any*) – Extra facets. For details, see *Extra Facets*.

#### Returns

Fixed cube.

#### Return type

`iris.cube.Cube`

`esmvalcore.cmor.fix.fix_file(file: Path, short_name: str, project: str, dataset: str, mip: str, output_dir: Path, add_unique_suffix: bool = False, session: Session | None = None, frequency: str | None = None, **extra_facets: Any) → Path | Sequence[Cube]`

Fix files before loading them into a `CubeList`.

This is mainly intended to fix errors that prevent loading the data with Iris (e.g., those related to `missing_value` or `_FillValue`) or operations that are more efficient with other packages (e.g., loading files with lots of variables is much faster with Xarray than Iris).

#### Parameters

- **file** (*Path*) – Path to the original file. Original files are not overwritten.
- **short\_name** (*str*) – Variable's short name.
- **project** (*str*) – Project of the dataset.
- **dataset** (*str*) – Name of the dataset.
- **mip** (*str*) – Variable's MIP.
- **output\_dir** (*Path*) – Output directory for fixed files.
- **add\_unique\_suffix** (*bool*) – Adds a unique suffix to `output_dir` for thread safety.
- **session** (*Session* | *None*) – Current session which includes configuration and directory information.
- **frequency** (*str* | *None*) – Variable's data frequency, if available.
- **\*\*extra\_facets** (*Any*) – Extra facets. For details, see *Extra Facets*.

#### Returns

Fixed data or a path to them.

**Return type**

Path | Sequence[Cube]

`esmvalcore.cmor.fix.fix_metadata(cubes: Sequence[Cube], short_name: str, project: str, dataset: str, mip: str, frequency: str | None = None, session: Session | None = None, **extra_facets: Any) → CubeList`

Fix cube metadata if fixes are required.

This method collects all the relevant fixes (including generic ones) for a given variable and applies them.

**Parameters**

- **cubes** (*Sequence[Cube]*) – Cubes to fix.
- **short\_name** (*str*) – Variable's short name.
- **project** (*str*) – Project of the dataset.
- **dataset** (*str*) – Name of the dataset.
- **mip** (*str*) – Variable's MIP.
- **frequency** (*str | None*) – Variable's data frequency, if available.
- **session** (*Session | None*) – Current session which includes configuration and directory information.
- **\*\*extra\_facets** (*Any*) – Extra facets. For details, see *Extra Facets*.

**Returns**

Fixed cubes.

**Return type**

`iris.cube.CubeList`

### 9.1.3 Functions for fixing issues

Functions for fixing specific issues with datasets.

**Functions:**

|                                           |                                                         |
|-------------------------------------------|---------------------------------------------------------|
| <code>add_altitude_from_plev(cube)</code> | Add altitude coordinate from pressure level coordinate. |
| <code>add_plev_from_altitude(cube)</code> | Add pressure level coordinate from altitude coordinate. |
| <code>get_next_month(month, year)</code>  | Get next month and year.                                |
| <code>get_time_bounds(time, freq)</code>  | Get bounds for time coordinate.                         |

`esmvalcore.cmor.fixes.add_altitude_from_plev(cube)`

Add altitude coordinate from pressure level coordinate.

**Parameters**

**cube** (*iris.cube.Cube*) – Input cube.

**Raises**

**ValueError** – cube does not contain coordinate `air_pressure`.

`esmvalcore.cmor.fixes.add_plev_from_altitude(cube)`

Add pressure level coordinate from altitude coordinate.

**Parameters**

**cube** (*iris.cube.Cube*) – Input cube.

**Raises**

**ValueError** – cube does not contain coordinate altitude.

`esmvalcore.cmor.fixes.get_next_month(month: int, year: int) → tuple[int, int]`

Get next month and year.

**Parameters**

- **month** (*int*) – Current month.
- **year** (*int*) – Current year.

**Returns**

Next month and next year.

**Return type**

`tuple[int, int]`

`esmvalcore.cmor.fixes.get_time_bounds(time: Coord, freq: str) → np.ndarray`

Get bounds for time coordinate.

For decadal data, use 1 January 5 years before/after the current year. For yearly data, use 1 January of the current year and 1 January of the next year. For monthly data, use the first day of the current month and the first day of the next month. For other frequencies (daily or *n*-hourly, where *n* is a divisor of 24), half of the frequency is subtracted/added from the current point in time to get the bounds.

**Parameters**

- **time** (*Coord*) – Time coordinate.
- **freq** (*str*) – Frequency.

**Returns**

Time bounds

**Return type**

`np.ndarray`

**Raises**

**NotImplementedError** – Non-supported frequency is given.

### 9.1.4 Using CMOR tables

CMOR information reader for ESMValTool.

Read variable information from CMOR 2 and CMOR 3 tables and make it easily available for the other components of ESMValTool

**Classes:**

|                                                             |                                                                       |
|-------------------------------------------------------------|-----------------------------------------------------------------------|
| <code>CMIP3Info</code> ([cmor_tables_path, default, ...])   | Class to read CMIP3-like CMOR tables.                                 |
| <code>CMIP5Info</code> ([cmor_tables_path, default, ...])   | Class to read CMIP5-like CMOR tables.                                 |
| <code>CMIP6Info</code> ([cmor_tables_path, default, ...])   | Class to read CMIP6-like CMOR tables.                                 |
| <code>CoordinateInfo</code> (name)                          | Class to read and store coordinate information.                       |
| <code>CustomInfo</code> ([cmor_tables_path])                | Class to read custom var info for ESMVal.                             |
| <code>InfoBase</code> ([default, alt_names, strict, paths]) | Base class for all CMOR table info classes.                           |
| <code>JsonInfo</code> ()                                    | Base class for the info classes.                                      |
| <code>NoInfo</code> ()                                      | Table that can be used for projects that do not provide a CMOR table. |
| <code>Obs4MIPsInfo</code> ([alt_names, strict, paths])      | Class to read obs4MIPs-like CMOR tables.                              |

continues on next page

Table 8 – continued from previous page

|                                                     |                                               |
|-----------------------------------------------------|-----------------------------------------------|
| <code>TableInfo(*args, **kwargs)</code>             | Container class for storing a CMOR table.     |
| <code>VariableInfo([table_type, short_name])</code> | Class to read and store variable information. |

**Data:**

|                          |                    |
|--------------------------|--------------------|
| <code>CMOR_TABLES</code> | CMOR info objects. |
|--------------------------|--------------------|

**Functions:**

|                                                            |                                                 |
|------------------------------------------------------------|-------------------------------------------------|
| <code>clear_table_cache()</code>                           | Clear the CMOR table cache.                     |
| <code>get_tables(session, project)</code>                  | Get the CMOR tables for a project.              |
| <code>get_var_info(project, mip, short_name[, ...])</code> | Get variable information.                       |
| <code>read_cmor_tables([cfg_developer])</code>             | Read cmor tables required in the configuration. |

```
class esmvalcore.cmor.table.CMIP3Info(cmor_tables_path: str | None = None, default: CustomInfo | None = None, alt_names: list[list[str]] | None = None, strict: bool = True, paths: Iterable[Path] = ())
```

Bases: `CMIP5Info`

Class to read CMIP3-like CMOR tables.

**Parameters**

- **cmor\_tables\_path** (`str` | `None`) – The path to a directory with subdirectory “Tables” where the CMOR tables are located.

Deprecated since version 2.14.0: The `cmor_tables_path` parameter is deprecated and will be removed in ESMValCore v2.16.0. Please use the `paths` parameter instead.

- **default** (`CustomInfo` | `None`) – Default table to look variables on if not found.

Deprecated since version 2.14.0: The `default` parameter is deprecated and will be removed in ESMValCore v2.16.0. Please use the `paths` parameter instead to aggregate multiple tables.

- **alt\_names** (`list[list[str]]` | `None`) – List of known alternative names for variables. If no value is provided, the default values from the installed copy of `variable_alt_names.yml` will be used.

- **strict** (`bool`) – If `False`, the function `get_variable()` will look for a variable in other tables if it can not be found in the table specified by `mip` in the `recipe` or `Dataset`.

- **default\_table\_prefix** – If the `table_id` contains a prefix, it can be specified here.

Deprecated since version 2.14.0: The `default_table_prefix` parameter is deprecated and will be removed in ESMValCore v2.16.0.

- **paths** (`Iterable[Path]`) – A list of paths to CMOR tables. The path can be relative to the built-in tables in the `esmvalcore/cmor/tables` directory, or any other path. The built-in tables will be used if the path is relative and exists in the built-in tables directory. Any file in the specified paths will be read as a CMOR table.

**Attributes:**

|                        |                                                                                                 |
|------------------------|-------------------------------------------------------------------------------------------------|
| <code>alt_names</code> | List of known alternative names for variables.                                                  |
| <code>coords</code>    | The coordinates defined in these tables.                                                        |
| <code>default</code>   | Default table to look variables on if not found.                                                |
| <code>paths</code>     | A list of paths to CMOR tables.                                                                 |
| <code>strict</code>    | If False, will look for a variable in other tables if it can not be found in the requested one. |
| <code>tables</code>    | A mapping from table names to <code>TableInfo</code> objects.                                   |

#### Methods:

|                                                             |                                             |
|-------------------------------------------------------------|---------------------------------------------|
| <code>get_table(table)</code>                               | Search and return the table info.           |
| <code>get_variable(table_name, short_name, *[, ...])</code> | Search and return the variable information. |

#### `alt_names`

List of known alternative names for variables.

#### `coords`: `dict[str, CoordinateInfo]`

The coordinates defined in these tables.

#### `default`

Default table to look variables on if not found.

Deprecated since version 2.14.0: The `default` attribute is deprecated and will be removed in ESMValCore v2.16.0.

#### `get_table(table: str) → TableInfo | None`

Search and return the table info.

##### Parameters

**table** (*str*) – Table name

##### Returns

Return the `TableInfo` object for the requested table if found, returns `None` if not

##### Return type

`TableInfo | None`

#### `get_variable(table_name: str, short_name: str, *, branding_suffix: str | None = None, derived: bool = False) → VariableInfo | None`

Search and return the variable information.

##### Parameters

- **table\_name** (*str*) – Table name, i.e., the mip in the *recipe* or *Dataset*, e.g. "Omon" for CMIP6 or "ocean" for CMIP7.
- **short\_name** (*str*) – Variable's short name, e.g. "tos" for sea surface temperature.
- **branding\_suffix** (*str | None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table, e.g. a *CMIP7 branding suffix*, could be "tavg-u-hxy-sea", which defines the temporal average at an undefined vertical level on a horizontal grid where non-sea points are masked.
- **derived** (*bool*) – Variable is derived. Information retrieval for derived variables always looks in the default tables (usually, the custom tables) if variable is not found in the requested table.

**Returns**

*VariableInfo* object for the requested variable if found, *None* otherwise.

**Return type**

*VariableInfo* | *None*

**paths**

A list of paths to CMOR tables.

**strict**

If *False*, will look for a variable in other tables if it can not be found in the requested one.

**tables:** `dict[str, TableInfo]`

A mapping from table names to *TableInfo* objects.

```
class esmvalcore.cmor.table.CMIP5Info(cmor_tables_path: str | None = None, default: CustomInfo | None = None, alt_names: list[list[str]] | None = None, strict: bool = True, paths: Iterable[Path] = ())
```

Bases: *InfoBase*

Class to read CMIP5-like CMOR tables.

This class reads CMOR 2 format tables.

**Parameters**

- **cmor\_tables\_path** (*str* | *None*) – The path to a directory with subdirectory “Tables” where the CMOR tables are located.  
  
Deprecated since version 2.14.0: The `cmor_tables_path` parameter is deprecated and will be removed in ESMValCore v2.16.0. Please use the `paths` parameter instead.
- **default** (*CustomInfo* | *None*) – Default table to look variables on if not found.  
  
Deprecated since version 2.14.0: The `default` parameter is deprecated and will be removed in ESMValCore v2.16.0. Please use the `paths` parameter instead to aggregate multiple tables.
- **alt\_names** (*list[list[str]]* | *None*) – List of known alternative names for variables. If no value is provided, the default values from the installed copy of `variable_alt_names.yml` will be used.
- **strict** (*bool*) – If *False*, the function `get_variable()` will look for a variable in other tables if it can not be found in the table specified by `mip` in the *recipe* or *Dataset*.
- **default\_table\_prefix** – If the `table_id` contains a prefix, it can be specified here.  
  
Deprecated since version 2.14.0: The `default_table_prefix` parameter is deprecated and will be removed in ESMValCore v2.16.0.
- **paths** (*Iterable[Path]*) – A list of paths to CMOR tables. The path can be relative to the built-in tables in the `esmvalcore/cmor/tables` directory, or any other path. The built-in tables will be used if the path is relative and exists in the built-in tables directory. Any file in the specified paths will be read as a CMOR table.

**Attributes:**

|                  |                                                  |
|------------------|--------------------------------------------------|
| <i>alt_names</i> | List of known alternative names for variables.   |
| <i>coords</i>    | The coordinates defined in these tables.         |
| <i>default</i>   | Default table to look variables on if not found. |

continues on next page

Table 13 – continued from previous page

|                     |                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------|
| <code>paths</code>  | A list of paths to CMOR tables.                                                                 |
| <code>strict</code> | If False, will look for a variable in other tables if it can not be found in the requested one. |
| <code>tables</code> | A mapping from table names to <i>TableInfo</i> objects.                                         |

**Methods:**

|                                                           |                                             |
|-----------------------------------------------------------|---------------------------------------------|
| <code>get_table(table)</code>                             | Search and return the table info.           |
| <code>get_variable(table_name, short_name, *, ...)</code> | Search and return the variable information. |

**alt\_names**

List of known alternative names for variables.

**coords:** `dict[str, CoordinateInfo]`

The coordinates defined in these tables.

**default**

Default table to look variables on if not found.

Deprecated since version 2.14.0: The `default` attribute is deprecated and will be removed in ESMValCore v2.16.0.

**get\_table**(*table: str*) → *TableInfo* | *None*

Search and return the table info.

**Parameters**

**table** (*str*) – Table name

**Returns**

Return the *TableInfo* object for the requested table if found, returns *None* if not

**Return type**

*TableInfo* | *None*

**get\_variable**(*table\_name: str, short\_name: str, \*, branding\_suffix: str | None = None, derived: bool = False*) → *VariableInfo* | *None*

Search and return the variable information.

**Parameters**

- **table\_name** (*str*) – Table name, i.e., the *mip* in the *recipe* or *Dataset*, e.g. "Omon" for CMIP6 or "ocean" for CMIP7.
- **short\_name** (*str*) – Variable's short name, e.g. "tos" for sea surface temperature.
- **branding\_suffix** (*str* | *None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table, e.g. a CMIP7 branding suffix, could be "tavg-u-hxy-sea", which defines the temporal average at an undefined vertical level on a horizontal grid where non-sea points are masked.
- **derived** (*bool*) – Variable is derived. Information retrieval for derived variables always looks in the default tables (usually, the custom tables) if variable is not found in the requested table.

**Returns**

*VariableInfo* object for the requested variable if found, *None* otherwise.

**Return type**

*VariableInfo* | None

**paths**

A list of paths to CMOR tables.

**strict**

If False, will look for a variable in other tables if it can not be found in the requested one.

**tables:** `dict[str, TableInfo]`

A mapping from table names to *TableInfo* objects.

```
class esmvalcore.cmor.table.CMIP6Info(cmor_tables_path: str | None = None, default: CustomInfo | None = None, alt_names: list[list[str]] | None = None, strict: bool = True, default_table_prefix: str = "", paths: Iterable[Path] = ())
```

Bases: *InfoBase*

Class to read CMIP6-like CMOR tables.

This class reads CMOR 3 json format tables.

**Parameters**

- **cmor\_tables\_path** (*str* | *None*) – The path to a directory with subdirectory “Tables” where the CMOR tables are located.

Deprecated since version 2.14.0: The `cmor_tables_path` parameter is deprecated and will be removed in ESMValCore v2.16.0. Please use the `paths` parameter instead.

- **default** (*CustomInfo* | *None*) – Default table to look variables on if not found.

Deprecated since version 2.14.0: The `default` parameter is deprecated and will be removed in ESMValCore v2.16.0. Please use the `paths` parameter instead to aggregate multiple tables.

- **alt\_names** (*list[list[str]]* | *None*) – List of known alternative names for variables. If no value is provided, the default values from the installed copy of `variable_alt_names.yml` will be used.

- **strict** (*bool*) – If `False`, the function `get_variable()` will look for a variable in other tables if it can not be found in the table specified by `mip` in the *recipe* or *Dataset*.

- **default\_table\_prefix** (*str*) – If the `table_id` contains a prefix, it can be specified here.

Deprecated since version 2.14.0: The `default_table_prefix` parameter is deprecated and will be removed in ESMValCore v2.16.0.

- **paths** (*Iterable[Path]*) – A list of paths to CMOR tables. The path can be relative to the built-in tables in the `esmvalcore/cmor/tables` directory, or any other path. The built-in tables will be used if the path is relative and exists in the built-in tables directory. Only files with the extension `.json` in the specified paths will be read as a CMOR tables, any other files will be ignored.

**Attributes:**

|                   |                                                                                          |
|-------------------|------------------------------------------------------------------------------------------|
| <i>activities</i> | A mapping from <code>exp</code> to <code>activity</code> from the controlled vocabulary. |
| <i>alt_names</i>  | List of known alternative names for variables.                                           |
| <i>coords</i>     | The coordinates defined in these tables.                                                 |
| <i>default</i>    | Default table to look variables on if not found.                                         |

continues on next page

Table 15 – continued from previous page

|                             |                                                                                                 |
|-----------------------------|-------------------------------------------------------------------------------------------------|
| <i>default_table_prefix</i> | If the <code>table_id</code> contains a prefix, it can be specified here.                       |
| <i>institutes</i>           | A mapping from dataset to institute from the controlled vocabulary.                             |
| <i>paths</i>                | A list of paths to CMOR tables.                                                                 |
| <i>strict</i>               | If False, will look for a variable in other tables if it can not be found in the requested one. |
| <i>tables</i>               | A mapping from table names to <i>TableInfo</i> objects.                                         |

**Methods:**

|                                                             |                                             |
|-------------------------------------------------------------|---------------------------------------------|
| <code>get_table(table)</code>                               | Search and return the table info.           |
| <code>get_variable(table_name, short_name, *[, ...])</code> | Search and return the variable information. |

**activities:** `dict[str, list[str]]`

A mapping from `exp` to `activity` from the controlled vocabulary.

**alt\_names**

List of known alternative names for variables.

**coords:** `dict[str, CoordinateInfo]`

The coordinates defined in these tables.

**default**

Default table to look variables on if not found.

Deprecated since version 2.14.0: The `default` attribute is deprecated and will be removed in ESMValCore v2.16.0.

**default\_table\_prefix**

If the `table_id` contains a prefix, it can be specified here.

Deprecated since version 2.14.0: The `default_table_prefix` attribute is deprecated and will be removed in ESMValCore v2.16.0.

**get\_table**(*table*: *str*) → *TableInfo* | *None*

Search and return the table info.

**Parameters**

**table** (*str*) – Table name

**Returns**

Return the *TableInfo* object for the requested table if found, returns *None* if not

**Return type**

*TableInfo* | *None*

**get\_variable**(*table\_name*: *str*, *short\_name*: *str*, \*, *branding\_suffix*: *str* | *None* = *None*, *derived*: *bool* = *False*) → *VariableInfo* | *None*

Search and return the variable information.

**Parameters**

- **table\_name** (*str*) – Table name, i.e., the `mip` in the *recipe* or *Dataset*, e.g. "Omon" for CMIP6 or "ocean" for CMIP7.
- **short\_name** (*str*) – Variable's short name, e.g. "tos" for sea surface temperature.

- **branding\_suffix** (*str* / *None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table, e.g. a CMIP7 branding suffix, could be "tavg-u-hxy-sea", which defines the temporal average at an undefined vertical level on a horizontal grid where non-sea points are masked.
- **derived** (*bool*) – Variable is derived. Information retrieval for derived variables always looks in the default tables (usually, the custom tables) if variable is not found in the requested table.

**Returns**

*VariableInfo* object for the requested variable if found, *None* otherwise.

**Return type**

*VariableInfo* | *None*

**institutes:** `dict[str, list[str]]`

A mapping from dataset to institute from the controlled vocabulary.

**paths**

A list of paths to CMOR tables.

**strict**

If *False*, will look for a variable in other tables if it can not be found in the requested one.

**tables:** `dict[str, TableInfo]`

A mapping from table names to *TableInfo* objects.

**var\_to\_freq:** `dict[str, dict[str, str]]`

```

esmvalcore.cmor.table.CMOR_TABLES: dict[str, InfoBase] = {'ACCESS':
CMIP6Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip6/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip6-custom')], strict=False, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'CESM':
CMIP6Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip6/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip6-custom')], strict=False, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'CMIP3':
CMIP3Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip3/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip3-custom')], strict=True, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'CMIP5':
CMIP5Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip5/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip5-custom')], strict=True, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'CMIP6':
CMIP6Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip6/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip6-custom')], strict=True, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'CMIP7':
CMIP6Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip7/tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip6-custom')], strict=True, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'CORDEX':
CMIP5Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cordex/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cordex-custom'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip5-custom')], strict=True, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'EMAC':
CMIP6Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip6/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip6-custom')], strict=False, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'ICON':
CMIP6Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip6/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip6-custom')], strict=False, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'IPSLCM':
CMIP6Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip6/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip6-custom')], strict=False, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'OBS':
CMIP5Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip5/Tables'),
PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/checkouts/
fix-pdf-build/esmvalcore/cmor/tables/cmip5-custom')], strict=False, alt_names=[['sic',
'siconc'], ['tro3', 'o3'], ['usi', 'siu'], ['vsi', 'siv']]), 'OBS6':
CMIP6Info(paths=[PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/esmvalcore/
checkouts/fix-pdf-build/esmvalcore/cmor/tables/cmip6/Tables'),

```

CMOR info objects.

Deprecated since version 2.14.0: The global CMOR\_TABLES dictionary is deprecated and will be removed in ESMValCore v2.16.0. Please use `get_tables()` to access the CMOR tables instead.

**Note**

If this dictionary is empty, it can be populated by loading the global configuration by importing the `esmvalcore.config` module.

**Type**

dict of str, obj

**class** `esmvalcore.cmor.table.CoordinateInfo`(*name*: str)

Bases: `JsonInfo`

Class to read and store coordinate information.

**Attributes:**

|                         |                                                 |
|-------------------------|-------------------------------------------------|
| <i>axis</i>             | Axis                                            |
| <i>generic_lev_name</i> | Generic level name                              |
| <i>long_name</i>        | Long name                                       |
| <i>must_have_bounds</i> | Whether bounds are required on this dimension   |
| <i>name</i>             | Name of the coordinate entry in the CMOR table. |
| <i>out_name</i>         | Out name                                        |
| <i>requested</i>        | Values requested                                |
| <i>standard_name</i>    | Standard name                                   |
| <i>stored_direction</i> | Direction in which the coordinate increases     |
| <i>units</i>            | Units                                           |
| <i>valid_max</i>        | Maximum allowed value                           |
| <i>valid_min</i>        | Minimum allowed value                           |
| <i>value</i>            | Coordinate value                                |
| <i>var_name</i>         | Short name                                      |

**Methods:**

|                                    |                                        |
|------------------------------------|----------------------------------------|
| <code>read_json</code> (json_data) | Read coordinate information from json. |
|------------------------------------|----------------------------------------|

**Parameters**

**name** (str)

**axis**

Axis

**generic\_lev\_coords:** dict[str, `CoordinateInfo`]

**generic\_lev\_name**

Generic level name

**long\_name**

Long name

**must\_have\_bounds**

Whether bounds are required on this dimension

**name**

Name of the coordinate entry in the CMOR table.

**out\_name**

Out name

This is the name of the variable in the file

**read\_json**(*json\_data*)

Read coordinate information from json.

Non-present options will be set to empty

**Parameters**

**json\_data** (*dict*) – dictionary created by the json reader containing coordinate information

**requested:** `list[str]`

Values requested

**standard\_name**

Standard name

**stored\_direction**

Direction in which the coordinate increases

**units**

Units

**valid\_max**

Maximum allowed value

**valid\_min**

Minimum allowed value

**value**

Coordinate value

**var\_name**

Short name

**class** `esmvalcore.cmor.table.CustomInfo`(*cmor\_tables\_path: str | Path | None = None*)

Bases: `CMIP5Info`

Class to read custom var info for ESMVal.

Deprecated since version 2.14.0: This class is deprecated and will be removed in ESMValCore v2.16.0. Please use `CMIP5Info` instead.

**Parameters**

**cmor\_tables\_path** (*str | Path | None*) – Full path to the table or name for the table if it is present in ESMValTool repository. If `None`, use default tables from `esmval-core/cmor/tables/custom`.

**Attributes:**

|                        |                                                                                                 |
|------------------------|-------------------------------------------------------------------------------------------------|
| <code>alt_names</code> | List of known alternative names for variables.                                                  |
| <code>coords</code>    | The coordinates defined in these tables.                                                        |
| <code>default</code>   | Default table to look variables on if not found.                                                |
| <code>paths</code>     | A list of paths to CMOR tables.                                                                 |
| <code>strict</code>    | If False, will look for a variable in other tables if it can not be found in the requested one. |
| <code>tables</code>    | A mapping from table names to <code>TableInfo</code> objects.                                   |

**Methods:**

|                                                             |                                      |
|-------------------------------------------------------------|--------------------------------------|
| <code>get_table(table)</code>                               | Search and return the table info.    |
| <code>get_variable(table_name, short_name, *[, ...])</code> | Search and return the variable info. |

**alt\_names**

List of known alternative names for variables.

**coords:** `dict[str, CoordinateInfo]`

The coordinates defined in these tables.

**default**

Default table to look variables on if not found.

Deprecated since version 2.14.0: The `default` attribute is deprecated and will be removed in ESMValCore v2.16.0.

**get\_table**(*table: str*) → `TableInfo` | `None`

Search and return the table info.

**Parameters**

**table** (*str*) – Table name

**Returns**

Return the `TableInfo` object for the requested table if found, returns `None` if not

**Return type**

`TableInfo` | `None`

**get\_variable**(*table\_name: str, short\_name: str, \*, branding\_suffix: str | None = None, derived: bool = False*) → `VariableInfo` | `None`

Search and return the variable info.

**Parameters**

- **table\_name** (*str*) – Table name, i.e., the mip in the *recipe* or *Dataset*, e.g. "Omon" for CMIP6 or "ocean" for CMIP7.
- **short\_name** (*str*) – Variable's short name, e.g. "tos" for sea surface temperature.
- **branding\_suffix** (*str* / `None`) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table, e.g. a `CMIP7` branding suffix, could be "tavg-u-hxy-sea", which defines the temporal average at an undefined vertical level on a horizontal grid where non-sea points are masked.
- **derived** (*bool*) – Variable is derived. Info retrieval for derived variables always looks on the default tables if variable is not found in the requested table. Ignored for custom tables.

**Returns**

*VariableInfo* object for the requested variable if found, returns None if not.

**Return type**

*VariableInfo* | None

**paths**

A list of paths to CMOR tables.

**strict**

If False, will look for a variable in other tables if it can not be found in the requested one.

**tables: dict[str, TableInfo]**

A mapping from table names to *TableInfo* objects.

**var\_to\_freq: dict[str, dict[str, str]]**

**class** `esmvalcore.cmor.table.InfoBase`(*default: CustomInfo | None = None, alt\_names: list[list[str]] | None = None, strict: bool = True, paths: Iterable[Path] = ()*)

Bases: `object`

Base class for all CMOR table info classes.

**Parameters**

- **default** (`CustomInfo` | `None`) – Default table to look variables on if not found.  
Deprecated since version 2.14.0: The `default` parameter is deprecated and will be removed in ESMValCore v2.16.0. Please use the `paths` parameter instead to aggregate multiple tables.
- **alt\_names** (`list[list[str]]` | `None`) – List of known alternative names for variables. If no value is provided, the default values from the installed copy of `variable_alt_names.yml` will be used.
- **strict** (`bool`) – If `False`, the function `get_variable()` will look for a variable in other tables if it can not be found in the table specified by `mip` in the `recipe` or `Dataset`.
- **paths** (`Iterable[Path]`) – A list of paths to CMOR tables. The path can be relative to the built-in tables in the `esmvalcore/cmor/tables` directory, or any other path. The built-in tables will be used if the path is relative and exists in the built-in tables directory.

**Attributes:**

|                        |                                                                                                               |
|------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>alt_names</code> | List of known alternative names for variables.                                                                |
| <code>coords</code>    | The coordinates defined in these tables.                                                                      |
| <code>default</code>   | Default table to look variables on if not found.                                                              |
| <code>paths</code>     | A list of paths to CMOR tables.                                                                               |
| <code>strict</code>    | If <code>False</code> , will look for a variable in other tables if it can not be found in the requested one. |
| <code>tables</code>    | A mapping from table names to <i>TableInfo</i> objects.                                                       |

**Methods:**

|                                                             |                                             |
|-------------------------------------------------------------|---------------------------------------------|
| <code>get_table(table)</code>                               | Search and return the table info.           |
| <code>get_variable(table_name, short_name, *[, ...])</code> | Search and return the variable information. |

**alt\_names**

List of known alternative names for variables.

**coords:** `dict[str, CoordinateInfo]`

The coordinates defined in these tables.

**default**

Default table to look variables on if not found.

Deprecated since version 2.14.0: The `default` attribute is deprecated and will be removed in ESMValCore v2.16.0.

**get\_table**(*table: str*) → *TableInfo* | *None*

Search and return the table info.

**Parameters**

**table** (*str*) – Table name

**Returns**

Return the *TableInfo* object for the requested table if found, returns *None* if not

**Return type**

*TableInfo*

**get\_variable**(*table\_name: str, short\_name: str, \*, branding\_suffix: str | None = None, derived: bool = False*) → *VariableInfo* | *None*

Search and return the variable information.

**Parameters**

- **table\_name** (*str*) – Table name, i.e., the *mip* in the *recipe* or *Dataset*, e.g. "Omon" for CMIP6 or "ocean" for CMIP7.
- **short\_name** (*str*) – Variable's short name, e.g. "tos" for sea surface temperature.
- **branding\_suffix** (*str | None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table, e.g. a CMIP7 branding suffix, could be "tavg-u-hxy-sea", which defines the temporal average at an undefined vertical level on a horizontal grid where non-sea points are masked.
- **derived** (*bool*) – Variable is derived. Information retrieval for derived variables always looks in the default tables (usually, the custom tables) if variable is not found in the requested table.

**Returns**

*VariableInfo* object for the requested variable if found, *None* otherwise.

**Return type**

*VariableInfo* | *None*

**paths**

A list of paths to CMOR tables.

**strict**

If *False*, will look for a variable in other tables if it can not be found in the requested one.

**tables:** `dict[str, TableInfo]`

A mapping from table names to *TableInfo* objects.

**class** `esmvalcore.cmor.table.JsonInfo`

Bases: `object`

Base class for the info classes.

Provides common utility methods to read json variables

**class** `esmvalcore.cmor.table.NoInfo`

Bases: `InfoBase`

Table that can be used for projects that do not provide a CMOR table.

**Attributes:**

|                        |                                                                                                 |
|------------------------|-------------------------------------------------------------------------------------------------|
| <code>alt_names</code> | List of known alternative names for variables.                                                  |
| <code>coords</code>    | The coordinates defined in these tables.                                                        |
| <code>default</code>   | Default table to look variables on if not found.                                                |
| <code>paths</code>     | A list of paths to CMOR tables.                                                                 |
| <code>strict</code>    | If False, will look for a variable in other tables if it can not be found in the requested one. |
| <code>tables</code>    | A mapping from table names to <code>TableInfo</code> objects.                                   |

**Methods:**

|                                                             |                                             |
|-------------------------------------------------------------|---------------------------------------------|
| <code>get_table(table)</code>                               | Search and return the table info.           |
| <code>get_variable(table_name, short_name, *[, ...])</code> | Search and return the variable information. |

**alt\_names**

List of known alternative names for variables.

**coords:** `dict[str, CoordinateInfo]`

The coordinates defined in these tables.

**default**

Default table to look variables on if not found.

Deprecated since version 2.14.0: The `default` attribute is deprecated and will be removed in ESMValCore v2.16.0.

**get\_table**(*table: str*) → `TableInfo` | `None`

Search and return the table info.

**Parameters**

**table** (*str*) – Table name

**Returns**

Return the `TableInfo` object for the requested table if found, returns `None` if not

**Return type**

`TableInfo`

**get\_variable**(*table\_name: str, short\_name: str, \*, branding\_suffix: str* | `None` = `None`, *derived: bool* = `False`) → `VariableInfo` | `None`

Search and return the variable information.

**Parameters**

- **table\_name** (*str*) – Table name, i.e., the mip in the *recipe* or *Dataset*, e.g. "Omon" for CMIP6 or "ocean" for CMIP7.
- **short\_name** (*str*) – Variable's short name, e.g. "tos" for sea surface temperature.
- **branding\_suffix** (*str* | *None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table, e.g. a [CMIP7 branding suffix](#), could be "tavg-u-hxy-sea", which defines the temporal average at an undefined vertical level on a horizontal grid where non-sea points are masked.
- **derived** (*bool*) – Variable is derived. Information retrieval for derived variables always looks in the default tables (usually, the custom tables) if variable is not found in the requested table.

**Returns**

*VariableInfo* object for the requested variable if found, *None* otherwise.

**Return type**

*VariableInfo* | *None*

**paths**

A list of paths to CMOR tables.

**strict**

If *False*, will look for a variable in other tables if it can not be found in the requested one.

**tables:** `dict[str, TableInfo]`

A mapping from table names to *TableInfo* objects.

```
class esmvalcore.cmor.table.Obs4MIPsInfo(alt_names: list[list[str]] | None = None, strict: bool = True,
 paths: Iterable[Path] = ())
```

Bases: [CMIP6Info](#)

Class to read obs4MIPs-like CMOR tables.

**Parameters**

- **alt\_names** (`list[list[str]]` | *None*) – List of known alternative names for variables. If no value is provided, the default values from the installed copy of `variable_alt_names.yml` will be used.
- **strict** (*bool*) – If *False*, the function `get_variable()` will look for a variable in other tables if it can not be found in the table specified by `mip` in the *recipe* or *Dataset*.
- **paths** (`Iterable[Path]`) – A list of paths to CMOR tables. The path can be relative to the built-in tables in the `esmvalcore/cmor/tables` directory, or any other path. The built-in tables will be used if the path is relative and exists in the built-in tables directory.

**Attributes:**

|                                   |                                                                                          |
|-----------------------------------|------------------------------------------------------------------------------------------|
| <code>activities</code>           | A mapping from <code>exp</code> to <code>activity</code> from the controlled vocabulary. |
| <code>alt_names</code>            | List of known alternative names for variables.                                           |
| <code>coords</code>               | The coordinates defined in these tables.                                                 |
| <code>default</code>              | Default table to look variables on if not found.                                         |
| <code>default_table_prefix</code> | If the <code>table_id</code> contains a prefix, it can be specified here.                |

continues on next page

Table 25 – continued from previous page

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>institutes</i>  | A mapping from dataset to <i>institute</i> from the controlled vocabulary.                      |
| <i>paths</i>       | A list of paths to CMOR tables.                                                                 |
| <i>strict</i>      | If False, will look for a variable in other tables if it can not be found in the requested one. |
| <i>tables</i>      | A mapping from table names to <i>TableInfo</i> objects.                                         |
| <i>var_to_freq</i> |                                                                                                 |

**Methods:**

|                                                        |                                             |
|--------------------------------------------------------|---------------------------------------------|
| <i>get_table</i> (table)                               | Search and return the table info.           |
| <i>get_variable</i> (table_name, short_name, *[, ...]) | Search and return the variable information. |

**activities:** `dict[str, list[str]]`

A mapping from *exp* to *activity* from the controlled vocabulary.

**alt\_names**

List of known alternative names for variables.

**coords:** `dict[str, CoordinateInfo]`

The coordinates defined in these tables.

**default**

Default table to look variables on if not found.

Deprecated since version 2.14.0: The `default` attribute is deprecated and will be removed in ESMValCore v2.16.0.

**default\_table\_prefix**

If the `table_id` contains a prefix, it can be specified here.

Deprecated since version 2.14.0: The `default_table_prefix` attribute is deprecated and will be removed in ESMValCore v2.16.0.

**get\_table**(table: *str*) → *TableInfo* | *None*

Search and return the table info.

**Parameters**

**table** (*str*) – Table name

**Returns**

Return the *TableInfo* object for the requested table if found, returns *None* if not

**Return type**

*TableInfo* | *None*

**get\_variable**(table\_name: *str*, short\_name: *str*, \*, branding\_suffix: *str* | *None* = *None*, derived: *bool* = *False*) → *VariableInfo* | *None*

Search and return the variable information.

**Parameters**

- **table\_name** (*str*) – Table name, i.e., the *mip* in the *recipe* or *Dataset*, e.g. "Omon" for CMIP6 or "ocean" for CMIP7.
- **short\_name** (*str*) – Variable's short name, e.g. "tos" for sea surface temperature.

- **branding\_suffix** (*str* / *None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table, e.g. a CMIP7 branding suffix, could be "tavg-u-hxy-sea", which defines the temporal average at an undefined vertical level on a horizontal grid where non-sea points are masked.
- **derived** (*bool*) – Variable is derived. Information retrieval for derived variables always looks in the default tables (usually, the custom tables) if variable is not found in the requested table.

**Returns**

*VariableInfo* object for the requested variable if found, *None* otherwise.

**Return type**

*VariableInfo* | *None*

**institutes:** `dict[str, list[str]]`

A mapping from dataset to institute from the controlled vocabulary.

**paths**

A list of paths to CMOR tables.

**strict**

If *False*, will look for a variable in other tables if it can not be found in the requested one.

**tables:** `dict[str, TableInfo]`

A mapping from table names to *TableInfo* objects.

**var\_to\_freq:** `dict[str, dict[str, str]]`

**class** `esmvalcore.cmor.table.TableInfo(*args, **kwargs)`

Bases: `dict`

Container class for storing a CMOR table.

**Methods:**

|                                          |                                                                                                                                                                                                                                                           |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>clear()</code>                     | Remove all items from the dict.                                                                                                                                                                                                                           |
| <code>copy()</code>                      | Return a shallow copy of the dict.                                                                                                                                                                                                                        |
| <code>fromkeys(iterable[, value])</code> | Create a new dictionary with keys from iterable and values set to value.                                                                                                                                                                                  |
| <code>get(key[, default])</code>         | Return the value for key if key is in the dictionary, else default.                                                                                                                                                                                       |
| <code>items()</code>                     | Return a set-like object providing a view on the dict's items.                                                                                                                                                                                            |
| <code>keys()</code>                      | Return a set-like object providing a view on the dict's keys.                                                                                                                                                                                             |
| <code>pop(k[,d])</code>                  | If the key is not found, return the default if given; otherwise, raise a <code>KeyError</code> .                                                                                                                                                          |
| <code>popitem()</code>                   | Remove and return a (key, value) pair as a 2-tuple.                                                                                                                                                                                                       |
| <code>setdefault(key[, default])</code>  | Insert key with a value of default if key is not in the dictionary.                                                                                                                                                                                       |
| <code>update([E,]**F)</code>             | If E is present and has a <code>.keys()</code> method, then does: for k in E.keys(): D[k] = E[k] If E is present and lacks a <code>.keys()</code> method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k] |
| <code>values()</code>                    | Return an object providing a view on the dict's values.                                                                                                                                                                                                   |

**Attributes:**

|                  |                               |
|------------------|-------------------------------|
| <i>frequency</i> | Table frequency (if defined). |
| <i>name</i>      | Table name.                   |
| <i>realm</i>     | Table realm (if defined).     |

**clear()**

Remove all items from the dict.

**copy()**

Return a shallow copy of the dict.

**frequency**

Table frequency (if defined).

**classmethod fromkeys**(*iterable, value=None, / (Positional-only parameter separator (PEP 570))*)

Create a new dictionary with keys from iterable and values set to value.

**get**(*key, default=None, /*)

Return the value for key if key is in the dictionary, else default.

**items()**

Return a set-like object providing a view on the dict's items.

**keys()**

Return a set-like object providing a view on the dict's keys.

**name**

Table name.

**pop**(*k[, d]*) → *v*, remove specified key and return the corresponding value.

If the key is not found, return the default if given; otherwise, raise a `KeyError`.

**popitem()**

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises `KeyError` if the dict is empty.

**realm**

Table realm (if defined).

**setdefault**(*key, default=None, /*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**update**(*[E, ]\*\*F*) → *None*. Update D from mapping/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E.keys(): D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()**

Return an object providing a view on the dict's values.

`class esmvalcore.cmor.table.VariableInfo(table_type: str = "", short_name: str = "")`

Bases: *JsonInfo*

Class to read and store variable information.

**Attributes:**

|                       |                        |
|-----------------------|------------------------|
| <i>coordinates</i>    | Coordinates            |
| <i>dimensions</i>     | List of dimensions     |
| <i>frequency</i>      | Data frequency         |
| <i>long_name</i>      | Long name              |
| <i>modeling_realm</i> | Modeling realm         |
| <i>positive</i>       | Increasing direction   |
| <i>short_name</i>     | Short name             |
| <i>standard_name</i>  | Standard name          |
| <i>units</i>          | Data units             |
| <i>valid_max</i>      | Maximum admitted value |
| <i>valid_min</i>      | Minimum admitted value |

**Methods:**

|                                                    |                                                                 |
|----------------------------------------------------|-----------------------------------------------------------------|
| <i>copy()</i>                                      | Return a shallow copy of VariableInfo.                          |
| <i>has_coord_with_standard_name(standard_name)</i> | Check if a coordinate with a given <i>standard_name</i> exists. |
| <i>read_json(json_data, default_freq)</i>          | Read variable information from json.                            |

**Parameters**

- **table\_type** (*str*)
- **short\_name** (*str*)

**coordinates:** `dict[str, CoordinateInfo]`

Coordinates

This is a dict with the names of the dimensions as keys and CoordinateInfo objects as values.

**copy()** → *Self*

Return a shallow copy of VariableInfo.

**Returns**

Shallow copy of this object.

**Return type**

*VariableInfo*

**dimensions:** `list[str]`

List of dimensions

**frequency**

Data frequency

**has\_coord\_with\_standard\_name(standard\_name: str)** → `bool`

Check if a coordinate with a given *standard\_name* exists.

For some coordinates, multiple (slightly different) versions with different dimension names but identical *standard\_name* exist. For example, the CMIP6 tables provide 4 different *standard\_name=time* dimensions:

*time*, *time1*, *time2*, and *time3*. Other examples would be the CMIP6 pressure levels (*plev19*, *plev23*, *plev27*, etc. with standard name *air\_pressure*) and the altitudes (*alt16*, *alt40* with standard name *altitude*).

This function can be used to check for the existence of a specific coordinate defined by its *standard\_name*, not its dimension name.

**Parameters**

**standard\_name** (*str*) – Standard name to be checked.

**Returns**

*True* if there is at least one coordinate with the given *standard\_name*, *False* if not.

**Return type**

`bool`

**long\_name**

Long name

**modeling\_realm:** `list[str]`

Modeling realm

**positive**

Increasing direction

**read\_json**(*json\_data: dict*, *default\_freq: str*) → `None`

Read variable information from json.

Non-present options will be set to empty

**Parameters**

- **json\_data** (*dict*) – Dictionary created by the json reader containing variable information.
- **default\_freq** (*str*) – Default frequency to use if it is not defined at variable level.

**Return type**

`None`

**short\_name**

Short name

**standard\_name**

Standard name

**units**

Data units

**valid\_max**

Maximum admitted value

**valid\_min**

Minimum admitted value

`esmvalcore.cmor.table.clear_table_cache()` → `None`

Clear the CMOR table cache.

**Return type**

`None`

`esmvalcore.cmor.table.get_tables(session: Session | Config, project: str) → InfoBase`

Get the CMOR tables for a project.

#### Parameters

- **session** (*Session* | *Config*) – The configuration.
- **project** (*str*) – The project to load a CMOR table for.

#### Return type

*InfoBase*

`esmvalcore.cmor.table.get_var_info(project: str, mip: str, short_name: str, branding_suffix: str | None = None) → VariableInfo | None`

Get variable information.

#### Note

If *project*=CORDEX and the *mip* ends with 'hr', it is cropped to 'h' since CORDEX X-hourly tables define the *mip* as ending in 'h' instead of 'hr'.

#### Parameters

- **project** (*str*) – Dataset's project.
- **mip** (*str*) – Variable's CMOR table, i.e., MIP.
- **short\_name** (*str*) – Variable's short name.
- **branding\_suffix** (*str* | *None*) – A suffix that will be appended to *short\_name* when looking up the variable in the CMOR table.

#### Returns

*VariableInfo* object for the requested variable if found, *None* otherwise.

#### Return type

*VariableInfo* | *None*

#### Raises

**KeyError** – No CMOR tables available for *project*.

`esmvalcore.cmor.table.read_cmor_tables(cfg_developer: Path | None = None) → None`

Read cmor tables required in the configuration.

Deprecated since version 2.14.0: The config-developer.yml file based configuration is deprecated and will no longer be supported in ESMValCore v2.16.0. Please use `load_cmor_tables()` instead of this function.

#### Parameters

**cfg\_developer** (*Path* | *None*) – Path to config-developer.yml file.

#### Raises

**TypeError** – If *cfg\_developer* is not a Path-like object

#### Return type

*None*

## 9.2 Configuration

This section describes the `config` module.

### 9.2.1 CFG

Configuration of ESMValCore/Tool is done via `CFG` object:

```
>>> from esmvalcore.config import CFG
>>> CFG
Config({'auxiliary_data_dir': PosixPath('/home/user/auxiliary_data'),
 'compress_netcdf': False,
 'config_developer_file': None,
 'drs': {'CMIP5': 'default', 'CMIP6': 'default'},
 'exit_on_warning': False,
 'log_level': 'info',
 'max_parallel_tasks': None,
 'output_dir': PosixPath('/home/user/esmvaltool_output'),
 'output_file_type': 'png',
 'profile_diagnostic': False,
 'remove_preproc_dir': True,
 'rootpath': {'CMIP5': '~/default_inputpath',
 'CMIP6': '~/default_inputpath',
 'default': '~/default_inputpath'},
 'save_intermediary_cubes': False})
```

All configuration parameters are listed [here](#).

`CFG` is essentially a python dictionary with a few extra functions, similar to `matplotlib.rcParams`. This means that values can be updated like this:

```
>>> CFG['output_dir'] = '~/esmvaltool_output'
>>> CFG['output_dir']
PosixPath('/home/user/esmvaltool_output')
```

Notice that `CFG` automatically converts the path to an instance of `pathlib.Path` and expands the home directory. All values entered into the config are validated to prevent mistakes, for example, it will warn you if you make a typo in the key:

```
>>> CFG['output_directory'] = '~/esmvaltool_output'
InvalidConfigParameter: `output_directory` is not a valid config parameter.
```

Or, if the value entered cannot be converted to the expected type:

```
>>> CFG['max_parallel_tasks'] = ''
InvalidConfigParameter: Key `max_parallel_tasks`: Could not convert '' to int
```

`CFG` is also flexible, so it tries to correct the type of your input if possible:

```
>>> CFG['max_parallel_tasks'] = '8' # str
>>> type(CFG['max_parallel_tasks'])
int
```

It is also possible to temporarily use different configuration options using the `context()` context manager:

```
>>> with CFG.context({"output_dir": "/path/to/output"}, log_level="debug"):
... print(CFG["output_dir"])
... print(CFG["log_level"])
PosixPath('/path/to/output')
debug
>>> print(CFG["output_dir"])
PosixPath('/home/user/esmvaltool_output')
>>> print(CFG["log_level"])
info
```

By default, the configuration is loaded from YAML files in the user's home directory at `~/.config/esmvaltool`. If set, this can be overwritten with the `ESMVALTOOL_CONFIG_DIR` environment variable. Defaults for options that are not specified explicitly are listed [here](#). To reload the current configuration object according to these rules, use:

```
>>> CFG.reload()
```

To load the configuration object from custom directories, use:

```
>>> dirs = ['my/default/config', 'my/custom/config']
>>> CFG.load_from_dirs(dirs)
```

To update the existing configuration object from custom directories, use:

```
>>> dirs = ['my/default/config', 'my/custom/config']
>>> CFG.update_from_dirs(dirs)
```

## 9.2.2 Session

Recipes and diagnostics will be run in their own directories. This behavior can be controlled via the `Session` object. A `Session` must always be initiated from the global `CFG` object:

```
>>> session = CFG.start_session(name='my_session')
```

A `Session` is very similar to the config. It is also a dictionary, and copies all the keys from the `CFG` object. At this moment, `session` is essentially a copy of `CFG`:

```
>>> print(session == CFG)
True
>>> session['output_dir'] = '~/my_output_dir'
>>> print(session == CFG) # False
False
```

A `Session` also knows about the directories where the data will be stored. The session name is used to prefix the directories.

```
>>> session.session_dir
/home/user/my_output_dir/my_session_20201203_155821
>>> session.run_dir
/home/user/my_output_dir/my_session_20201203_155821/run
>>> session.work_dir
/home/user/my_output_dir/my_session_20201203_155821/work
>>> session.preproc_dir
/home/user/my_output_dir/my_session_20201203_155821/preproc
```

(continues on next page)

(continued from previous page)

```
>>> session.plot_dir
/home/user/my_output_dir/my_session_20201203_155821/plots
```

Unlike the global configuration, of which only one can exist, multiple sessions can be initiated from *CFG*.

### 9.2.3 API reference

Configuration module.

`esmvalcore.config.CFG`

Global ESMValCore configuration object of type *esmvalcore.config.Config*.

By default, this will be loaded from YAML files in the user configuration directory (by default `~/.config/esmvaltool`, but this can be changed with the `ESMVALTOOL_CONFIG_DIR` environment variable) similar to the way *Dask* handles configuration.

**Classes:**

|                                 |                                                                      |
|---------------------------------|----------------------------------------------------------------------|
| <i>Config</i> (*args, **kwargs) | ESMValTool configuration object.                                     |
| <i>Session</i> (config[, name]) | Container class for session configuration and directory information. |

**class** `esmvalcore.config.Config(*args, **kwargs)`

ESMValTool configuration object.

Do not instantiate this class directly, but use *esmvalcore.config.CFG* instead.

**Methods:**

|                                    |                                                                             |
|------------------------------------|-----------------------------------------------------------------------------|
| <i>clear</i> ()                    | Clear contents of configuration object.                                     |
| <i>context</i> ([mapping])         | Set configuration options temporarily inside a <code>with</code> statement. |
| <i>copy</i> ()                     | Copy the keys/values of this object to a dict.                              |
| <i>load_from_dirs</i> (dirs)       | Clear and load configuration object from directories.                       |
| <i>nested_update</i> (new_options) | Nested update of configuration object with another mapping.                 |
| <i>reload</i> ()                   | Clear and reload the configuration object.                                  |
| <i>start_session</i> (name)        | Start a new session from this configuration object.                         |
| <i>update_from_dirs</i> (dirs)     | Update configuration object from directories.                               |

**clear()** → *None*

Clear contents of configuration object.

**Return type**

*None*

**context**(*mapping*: *Mapping* | *None* = *None*, \*\*kwargs: *Any*) → *Generator*[*None*, *None*, *None*]

Set configuration options temporarily inside a `with` statement.

This configuration will only be effective inside the context manager.

**Parameters**

- **mapping** (*Mapping* | *None*) – Mapping with temporary configuration options.

- **\*\*kwargs** (*Any*) – Temporary configuration options.

**Return type**

Generator[None, None, None]

**copy()** → dict[str, Any]

Copy the keys/values of this object to a dict.

**Return type**

dict[str, Any]

**load\_from\_dirs**(*dirs: Iterable[str | Path]*) → None

Clear and load configuration object from directories.

This searches for all YAML files within the given directories and merges them together using `dask.config.collect()`. Nested objects are properly considered; see `dask.config.update()` for details. Values in the latter directories are preferred to those in the former.

Options that are not explicitly specified via YAML files are set to the *default values*.

**Note**

Just like `dask.config.collect()`, this silently ignores non-existing directories.

**Parameters**

**dirs** (*Iterable[str | Path]*) – A list of directories to search for YAML configuration files.

**Raises**

`esmvalcore.exceptions.InvalidConfigParameter` – Invalid configuration option given.

**Return type**

None

**nested\_update**(*new\_options: Mapping*) → None

Nested update of configuration object with another mapping.

Merge the existing configuration object with a new mapping using `dask.config.merge()` (new values are preferred over old values). Nested objects are properly considered; see `dask.config.update()` for details.

**Parameters**

**new\_options** (*Mapping*) – New configuration options.

**Raises**

`esmvalcore.exceptions.InvalidConfigParameter` – Invalid configuration option given.

**Return type**

None

**reload()** → None

Clear and reload the configuration object.

This will read all YAML files in the user configuration directory (by default `~/.config/esmvaltool`, but this can be changed with the `ESMVALTOOL_CONFIG_DIR` environment variable) and merges them together using `dask.config.collect()`. Nested objects are properly considered; see `dask.config.update()` for details.

Options that are not explicitly specified via YAML files are set to the *default values*.

**Note**

If the user configuration directory does not exist, this will be silently ignored.

**Raises**

`esmvalcore.exceptions.InvalidConfigParameter` – Invalid configuration option given.

**Return type**

None

`start_session(name: str) → Session`

Start a new session from this configuration object.

**Parameters**

**name** (*str*) – Name of the session.

**Return type**

*Session*

`update_from_dirs(dirs: Iterable[str | Path]) → None`

Update configuration object from directories.

This will first search for all YAML files within the given directories and merge them together using `dask.config.collect()` (if identical values are provided in multiple files, the value from the last file will be used). Then, the current configuration is merged with these new configuration options using `dask.config.merge()` (new values are preferred over old values). Nested objects are properly considered; see `dask.config.update()` for details.

**Note**

Just like `dask.config.collect()`, this silently ignores non-existing directories.

**Parameters**

**dirs** (*Iterable[str | Path]*) – A list of directories to search for YAML configuration files.

**Raises**

`esmvalcore.exceptions.InvalidConfigParameter` – Invalid configuration option given.

**Return type**

None

`class esmvalcore.config.Session(config: dict, name: str = 'session')`

Container class for session configuration and directory information.

Do not instantiate this class directly, but use `CFG.start_session` instead.

**Parameters**

- **config** (*dict*) – Dictionary with configuration settings.
- **name** (*str*) – Name of the session to initialize, for example, the name of the recipe (default='session').

**Methods:**

|                                       |                                                                             |
|---------------------------------------|-----------------------------------------------------------------------------|
| <code>clear()</code>                  | Clear contents of configuration object.                                     |
| <code>context([mapping])</code>       | Set configuration options temporarily inside a <code>with</code> statement. |
| <code>copy()</code>                   | Copy the keys/values of this object to a dict.                              |
| <code>set_session_name([name])</code> | Set the name for the session.                                               |

**Attributes:**

|                                      |                                                                                                           |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------|
| <code>cmor_log</code>                | Path to the log file with CMOR check messages.                                                            |
| <code>main_log</code>                | Path to the log file.                                                                                     |
| <code>main_log_debug</code>          | Path to the debug log file.                                                                               |
| <code>plot_dir</code>                | Directory with diagnostic script plot files.                                                              |
| <code>preproc_dir</code>             | Directory with preprocessor output files.                                                                 |
| <code>relative_cmor_log</code>       | Relative path to the log file with CMOR check messages, with respect to <code>session_dir</code> .        |
| <code>relative_main_log</code>       | Relative path to the log file, with respect to <code>session_dir</code> .                                 |
| <code>relative_main_log_debug</code> | Relative path to the debug log file, with respect to <code>session_dir</code> .                           |
| <code>relative_plot_dir</code>       | Relative path to diagnostic script plot directory, with respect to <code>session_dir</code> .             |
| <code>relative_preproc_dir</code>    | Relative path to the preprocessor output directory, with respect to <code>session_dir</code> .            |
| <code>relative_run_dir</code>        | Relative path to the directory with information about the run, with respect to <code>session_dir</code> . |
| <code>relative_work_dir</code>       | Relative path to diagnostic script output directory, with respect to <code>session_dir</code> .           |
| <code>run_dir</code>                 | Directory containing information about the run.                                                           |
| <code>session_dir</code>             | Session directory.                                                                                        |
| <code>work_dir</code>                | Directory with diagnostic script output files.                                                            |

`clear()` → `None`

Clear contents of configuration object.

**Return type**

`None`

**property `cmor_log`**

Path to the log file with CMOR check messages.

**context** (*mapping*: *Mapping* | *None* = *None*, *\*\*kwargs*: *Any*) → `Generator[None, None, None]`

Set configuration options temporarily inside a `with` statement.

This configuration will only be effective inside the context manager.

**Parameters**

- **mapping** (*Mapping* | *None*) – Mapping with temporary configuration options.
- **\*\*kwargs** (*Any*) – Temporary configuration options.

**Return type**

`Generator[None, None, None]`

**copy()** → dict[str, Any]

Copy the keys/values of this object to a dict.

**Return type**

dict[str, Any]

**property main\_log**

Path to the log file.

**property main\_log\_debug**

Path to the debug log file.

**property plot\_dir**

Directory with diagnostic script plot files.

**property preproc\_dir**

Directory with preprocessor output files.

**relative\_cmor\_log = PosixPath('run/cmor\_log.txt')**

Relative path to the log file with CMOR check messages, with respect to *session\_dir*.

**relative\_main\_log = PosixPath('run/main\_log.txt')**

Relative path to the log file, with respect to *session\_dir*.

**relative\_main\_log\_debug = PosixPath('run/main\_log\_debug.txt')**

Relative path to the debug log file, with respect to *session\_dir*.

**relative\_plot\_dir = PosixPath('plots')**

Relative path to diagnostic script plot directory, with respect to *session\_dir*.

**relative\_preproc\_dir = PosixPath('preproc')**

Relative path to the preprocessor output directory, with respect to *session\_dir*.

**relative\_run\_dir = PosixPath('run')**

Relative path to the directory with information about the run, with respect to *session\_dir*.

**relative\_work\_dir = PosixPath('work')**

Relative path to diagnostic script output directory, with respect to *session\_dir*.

**property run\_dir**

Directory containing information about the run.

**property session\_dir**

Session directory.

This is a uniquely named directory inside the *output directory*.

**session\_name: str | None**

**set\_session\_name**(name: str = 'session') → None

Set the name for the session.

The *name* is used to name the session directory, e.g. *session\_20201208\_132800/*. The date is suffixed automatically.

**Parameters**

**name** (*str*)

**Return type**

None

**property work\_dir**

Directory with diagnostic script output files.

## 9.3 Dataset

Classes and functions for defining, finding, and loading data.

**Classes:**

|                                |                                                         |
|--------------------------------|---------------------------------------------------------|
| <code>Dataset(**facets)</code> | Define datasets, find the related files, and load them. |
|--------------------------------|---------------------------------------------------------|

**Data:**

|                               |                   |
|-------------------------------|-------------------|
| <code>INHERITED_FACETS</code> | Inherited facets. |
|-------------------------------|-------------------|

**Functions:**

|                                                     |                                          |
|-----------------------------------------------------|------------------------------------------|
| <code>datasets_to_recipe(datasets[, recipe])</code> | Create or update a recipe from datasets. |
|-----------------------------------------------------|------------------------------------------|

**class** `esmvalcore.dataset.Dataset(**facets: FacetValue)`

Define datasets, find the related files, and load them.

**Parameters**

**\*\*facets** (*FacetValue*) – Facets describing the dataset. See *Facets* for the mapping between the facet names used by ESMValCore and those used on ESGF and *Variables and CMOR Tables* to find out which variables are available.

**supplementaries**

List of supplementary datasets.

**Type**

`list[Dataset]`

**facets**

Facets describing the dataset.

**Type**

`esmvalcore.typing.Facets`

**Methods:**

|                                               |                                                      |
|-----------------------------------------------|------------------------------------------------------|
| <code>add_supplementary(**facets)</code>      | Add an supplementary dataset.                        |
| <code>augment_facets()</code>                 | Add additional facets.                               |
| <code>copy(**facets)</code>                   | Create a copy.                                       |
| <code>find_files()</code>                     | Find files.                                          |
| <code>from_files()</code>                     | Create datasets based on the available files.        |
| <code>from_ranges()</code>                    | Create a list of datasets from short notations.      |
| <code>from_recipe(recipe, session)</code>     | Read datasets from a recipe.                         |
| <code>load()</code>                           | Load dataset.                                        |
| <code>set_facet(key, value[, persist])</code> | Set facet.                                           |
| <code>set_version()</code>                    | Set the 'version' facet based on the available data. |

continues on next page

Table 38 – continued from previous page

|                                  |                                   |
|----------------------------------|-----------------------------------|
| <code>summary</code> ([shorten]) | Summarize the content of dataset. |
|----------------------------------|-----------------------------------|

**Attributes:**

|                             |                                                                       |
|-----------------------------|-----------------------------------------------------------------------|
| <code>files</code>          | The files associated with this dataset.                               |
| <code>minimal_facets</code> | Return a dictionary with the persistent facets.                       |
| <code>session</code>        | A <code>esmvalcore.config.Session</code> associated with the dataset. |

**add\_supplementary**(\*\*facets: *FacetValue*) → None

Add an supplementary dataset.

This is a convenience function that will create a copy of the current dataset, update its facets with the values specified in **\*\*facets**, and append it to `Dataset.supplementaries`. For more control over the creation of the supplementary dataset, first create a new `Dataset` describing the supplementary dataset and then append it to `Dataset.supplementaries`.

**Parameters**

**\*\*facets** (*FacetValue*) – Facets describing the supplementary variable.

**Return type**

None

**augment\_facets**() → None

Add additional facets.

This function will update the dataset with additional facets from various sources. These include *Extra Facets* as well as facets read from the controlled vocabulary included in the CMOR tables if applicable.

**Return type**

None

**copy**(\*\*facets: *FacetValue*) → *Dataset*

Create a copy.

**Parameters**

**\*\*facets** (*FacetValue*) – Update these facets in the copy. Note that for supplementary datasets attached to the dataset, the 'short\_name' and 'mip' facets will not be updated with these values.

**Returns**

A copy of the dataset.

**Return type**

*Dataset*

**property files:** `list[DataElement]`

The files associated with this dataset.

**find\_files**() → None

Find files.

Look for files and populate the `Dataset.files` property of the dataset and its supplementary datasets.

**Return type**

None

**from\_files()** → Iterator[*Dataset*]

Create datasets based on the available files.

The facet values for local files are retrieved from the directory tree where the directories represent the facets values. See *CMIP data* for more information on this kind of file organization.

`glob.glob()` patterns can be used as facet values to select multiple datasets. If for some of the datasets not all glob patterns can be expanded (e.g. because the required facet values cannot be inferred from the directory names), these datasets will be ignored, unless this happens to be all datasets.

If `glob.glob()` patterns are used in supplementary variables and multiple matching datasets are found, only the supplementary dataset that has most facets in common with the main dataset will be attached.

Supplementary datasets will inherit the facet values from the main dataset for those facets listed in *INHERITED\_FACETS*.

### Examples

See *Discovering data* notebook for example use cases.

#### Yields

*Dataset* – Datasets representing the available files.

#### Return type

Iterator[*Dataset*]

**from\_ranges()** → list[*Dataset*]

Create a list of datasets from short notations.

This expands the 'ensemble' and 'sub\_experiment' facets in the dataset definition if they are ranges.

For example 'ensemble'='r(1:3)i1p1f1' will be expanded to three datasets, with 'ensemble' values 'r1i1p1f1', 'r2i1p1f1', 'r3i1p1f1'.

#### Returns

The datasets.

#### Return type

list[*Dataset*]

**static from\_recipe(recipe: Path | str | dict, session: Session) → list[*Dataset*]**

Read datasets from a recipe.

#### Parameters

- **recipe** (*Path* | *str* | *dict*) – *Recipe* to load the datasets from. The value provided here should be either a path to a file, a recipe file that has been loaded using e.g. `yaml.safe_load()`, or an *str* that can be loaded using `yaml.safe_load()`.
- **session** (*Session*) – Datasets to use in the recipe.

#### Returns

A list of datasets.

#### Return type

list[*Dataset*]

**load()** → Cube

Load dataset.

#### Raises

*InputFilesNotFound* – When no files were found.

**Returns**

An `iris` cube with the data corresponding the the dataset.

**Return type**

`iris.cube.Cube`

**property minimal\_facets: Facets**

Return a dictionary with the persistent facets.

**property session: Session**

A `esmvalcore.config.Session` associated with the dataset.

**set\_facet(key: str, value: FacetValue, persist: bool = True) → None**

Set facet.

**Parameters**

- **key** (`str`) – The name of the facet.
- **value** (`FacetValue`) – The value of the facet.
- **persist** (`bool`) – When writing a dataset to a recipe, only persistent facets will get written.

**Return type**

`None`

**set\_version() → None**

Set the 'version' facet based on the available data.

**Return type**

`None`

**summary(shorten: bool = False) → str**

Summarize the content of dataset.

**Parameters**

**shorten** (`bool`) – Shorten the summary.

**Returns**

A summary describing the dataset.

**Return type**

`str`

`esmvalcore.dataset.INHERITED_FACETS: list[str] = ['dataset', 'region', 'domain', 'driver', 'grid', 'project', 'timerange']`

Inherited facets.

Supplementary datasets created based on the available files using the `Dataset.from_files()` method will inherit the values of these facets from the main dataset.

`esmvalcore.dataset.datasets_to_recipe(datasets: Iterable[Dataset], recipe: Path | str | dict[str, Any] | None = None) → dict`

Create or update a recipe from datasets.

**Parameters**

- **datasets** (`Iterable[Dataset]`) – Datasets to use in the recipe.
- **recipe** (`Path | str | dict[str, Any] | None`) – *Recipe* to load the datasets from. The value provided here should be either a path to a file, a recipe file that has been loaded using e.g. `yaml.safe_load()`, or an `str` that can be loaded using `yaml.safe_load()`.

**Return type**

dict

**Examples**

See *Composing recipes* notebook for example use cases.

**Returns**

The recipe with the datasets. To convert the `dict` to a *recipe*, use e.g. `yaml.safe_dump()`.

**Return type**

dict

**Raises**

*RecipeError* – Raised when a dataset is missing the diagnostic facet.

**Parameters**

- **datasets** (*Iterable*[*Dataset*])
- **recipe** (*Path* | *str* | *dict*[*str*, *Any*] | *None*)

## 9.4 Find and download files from ESGF

### 9.4.1 esmvalcore.esgf

Find files on the ESGF and download them.

Deprecated since version 2.14.0: This module has been moved to *esmvalcore.io.esgf*. Importing it as `esmvalcore.esgf` is deprecated and will be removed in version 2.16.0.

**Classes:**

|                                              |                   |
|----------------------------------------------|-------------------|
| ESGFDataSource(name, project, priority, ...) |                   |
| ESGFFile(results[, dest_folder])             | File on the ESGF. |

**Functions:**

|                                                               |                                          |
|---------------------------------------------------------------|------------------------------------------|
| <code>download(files[, dest_folder, n_jobs])</code>           | Download multiple ESGFFiles in parallel. |
| <code>find_files(*, project, short_name, dataset, ...)</code> | Search for files on ESGF.                |

**class** `esmvalcore.esgf.ESGFDataSource`(*name*: 'str', *project*: 'str', *priority*: 'int', *download\_dir*: 'Path')

Bases: *DataSource*

**Attributes:**

|                           |                                                              |
|---------------------------|--------------------------------------------------------------|
| <code>debug_info</code>   | A string containing debug information when no data is found. |
| <code>download_dir</code> | The destination directory where data will be downloaded.     |
| <code>name</code>         | A name identifying the data source.                          |
| <code>priority</code>     | The priority of the data source.                             |
| <code>project</code>      | The project that the data source provides data for.          |

**Methods:**

|                                  |            |
|----------------------------------|------------|
| <code>find_data(**facets)</code> | Find data. |
|----------------------------------|------------|

**Parameters**

- **name** (*str*)
- **project** (*str*)
- **priority** (*int*)
- **download\_dir** (*Path*)

**debug\_info:** *str* = ''

A string containing debug information when no data is found.

**download\_dir:** *Path*

The destination directory where data will be downloaded.

**find\_data**(\*\**facets*: *FacetValue*) → *list*[*ESGFFile*]

Find data.

**Parameters****\*\*facets** (*FacetValue*) – Find data matching these facets.**Returns**

A list of files that have been found on ESGF.

**Return type***list* of *esmvalcore.io.esgf.ESGFFile***name:** *str*

A name identifying the data source.

**priority:** *int*

The priority of the data source. Lower values have priority.

**project:** *str*

The project that the data source provides data for.

**class** *esmvalcore.esgf.ESGFFile*(*results*: *Iterable*[*FileResult*], *dest\_folder*: *Path* | *None* = *None*)Bases: *DataElement*

File on the ESGF.

This is the object returned by *esmvalcore.io.esgf.find\_files()*.**Parameters**

- **results** (*Iterable*[*FileResult*])
- **dest\_folder** (*Path* | *None*)

**dataset**

The name of the dataset that the file is part of.

**Type***str***facets**

Facets describing the file.

**Type**  
dict[str,str]

**name**  
The name of the file.

**Type**  
str

**size**  
The size of the file in bytes.

**Type**  
int

**urls**  
The URLs where the file can be downloaded.

**Type**  
list[str]

**Attributes:**

|            |                                                     |
|------------|-----------------------------------------------------|
| attributes | Attributes are key-value pairs describing the data. |
|------------|-----------------------------------------------------|

**Methods:**

|                         |                                                   |
|-------------------------|---------------------------------------------------|
| download(dest_folder)   | Download the file.                                |
| local_file(dest_folder) | Return the path to the local file after download. |
| prepare()               | Prepare the data for access.                      |
| to_iris()               | Load the data as Iris cubes.                      |

**property attributes:** dict[str, Any]  
Attributes are key-value pairs describing the data.

**download**(dest\_folder: Path | None) → LocalFile  
Download the file.

**Parameters**  
**dest\_folder** (Path | None) – The destination folder.

**Raises**  
**DownloadError:** – Raised if downloading the file failed.

**Returns**  
The path where the file will be located after download.

**Return type**  
LocalFile

**local\_file**(dest\_folder: Path | None) → LocalFile  
Return the path to the local file after download.

**Parameters**  
**dest\_folder** (Path | None) – The destination folder.

**Returns**  
The path where the file will be located after download.

**Return type**  
LocalFile

**prepare()** → None

Prepare the data for access.

**Return type**  
None

**to\_iris()** → iris.cube.CubeList

Load the data as Iris cubes.

**Returns**  
The loaded data.

**Return type**  
iris.cube.CubeList

esmvalcore.esgf.**download**(files, dest\_folder=None, n\_jobs=4)

Download multiple ESGFFiles in parallel.

**Parameters**

- **files** (list of ESGFFile) – The files to download.
- **dest\_folder** (*Path or None*) – The destination folder.
- **n\_jobs** (*int*) – The number of files to download in parallel.

**Raises**

**DownloadError:** – Raised if one or more files failed to download.

esmvalcore.esgf.**find\_files**(\*, project, short\_name, dataset, \*\*facets)

Search for files on ESGF.

**Parameters**

- **project** (*str*) – Choose from CMIP3, CMIP5, CMIP6, CORDEX, or obs4MIPs.
- **short\_name** (*str*) – The name of the variable.
- **dataset** (*str*) – The name of the dataset.
- **\*\*facets** (*Union[str, list[str]]*) – Any other search facets. An '\*' can be used to match any value. By default, only the latest version of a file will be returned. To select all versions use `version='*'` while other omitted facets will default to '\*'. It is also possible to specify multiple values for a facet, e.g. `exp=['historical', 'ssp585']` will match any file that belongs to either the historical or ssp585 experiment. The `timerange` facet can be specified in [ISO 8601 format](#).

**Note**

A value of `timerange='*'` is supported, but combining a '\*' with a time or period *as supported in the recipe* is currently not supported and will return all found files.

**Examples**

Examples of how to use this function for all supported projects.

Search for a CMIP3 dataset:

```
>>> find_files(
... project='CMIP3',
... frequency='mon',
... short_name='tas',
... dataset='cccma_cgcm3_1',
... exp='historical',
... ensemble='run1',
...)
[ESGFFile:cmip3/CCCma/cccma_cgcm3_1/historical/mon/atmos/run1/tas/v1/tas_a1_20c3m_1_
↪cgcm3.1_t47_1850_2000.nc]
```

Search for a CMIP5 dataset:

```
>>> find_files(
... project='CMIP5',
... mip='Amon',
... short_name='tas',
... dataset='inmcm4',
... exp='historical',
... ensemble='r1i1p1',
...)
[ESGFFile:cmip5/output1/INM/inmcm4/historical/mon/atmos/Amon/r1i1p1/v20130207/tas_
↪Amon_inmcm4_historical_r1i1p1_185001-200512.nc]
```

Search for a CMIP6 dataset:

```
>>> find_files(
... project='CMIP6',
... mip='Amon',
... short_name='tas',
... dataset='CanESM5',
... exp='historical',
... ensemble='r1i1p1f1',
...)
[ESGFFile:CMIP6/CMIP/CCCma/CanESM5/historical/r1i1p1f1/Amon/tas/gn/v20190429/tas_
↪Amon_CanESM5_historical_r1i1p1f1_gn_185001-201412.nc]
```

Search for a CORDEX dataset and limit the search results to files containing data to the years in the range 1990-2000:

```
>>> find_files(
... project='CORDEX',
... frequency='mon',
... dataset='COSMO-crCLIM-v1-1',
... short_name='tas',
... exp='historical',
... ensemble='r1i1p1',
... domain='EUR-11',
... driver='MPI-M-MPI-ESM-LR',
... timerange='1990/2000',
...)
[ESGFFile:cordex/output/EUR-11/CLMcom-ETH/MPI-M-MPI-ESM-LR/historical/r1i1p1/COSMO-
↪crCLIM-v1-1/v1/mon/tas/v20191219/tas_EUR-11_MPI-M-MPI-ESM-LR_historical_r1i1p1_
```

(continues on next page)

(continued from previous page)

```

↪CLMcom-ETH-COSMO-crCLIM-v1-1_v1_mon_198101-199012.nc,
ESGFFile:cordex/output/EUR-11/CLMcom-ETH/MPI-M-MPI-ESM-LR/historical/r1i1p1/COSMO-
↪crCLIM-v1-1/v1/mon/tas/v20191219/tas_EUR-11_MPI-M-MPI-ESM-LR_historical_r1i1p1_
↪CLMcom-ETH-COSMO-crCLIM-v1-1_v1_mon_199101-200012.nc]

```

Search for an obs4MIPs dataset:

```

>>> find_files(
... project='obs4MIPs',
... frequency='mon',
... dataset='CERES-EBAF',
... short_name='rsutcs',
...)
[ESGFFile:obs4MIPs/NASA-LaRC/CERES-EBAF/atmos/mon/v20160610/rsutcs_CERES-EBAF_L3B_
↪Ed2-8_200003-201404.nc]

```

Search for any ensemble member:

```

>>> find_files(
... project='CMIP6',
... mip='Amon',
... short_name='tas',
... dataset='BCC-CSM2-MR',
... exp='historical',
... ensemble='*',
...)
[ESGFFile:CMIP6/CMIP/BCC/BCC-CSM2-MR/historical/r1i1p1f1/Amon/tas/gn/v20181126/tas_
↪Amon_BCC-CSM2-MR_historical_r1i1p1f1_gn_185001-201412.nc,
ESGFFile:CMIP6/CMIP/BCC/BCC-CSM2-MR/historical/r2i1p1f1/Amon/tas/gn/v20181115/tas_
↪Amon_BCC-CSM2-MR_historical_r2i1p1f1_gn_185001-201412.nc,
ESGFFile:CMIP6/CMIP/BCC/BCC-CSM2-MR/historical/r3i1p1f1/Amon/tas/gn/v20181119/tas_
↪Amon_BCC-CSM2-MR_historical_r3i1p1f1_gn_185001-201412.nc]

```

Search for all available versions of a file:

```

>>> find_files(
... project='CMIP5',
... mip='Amon',
... short_name='tas',
... dataset='CCSM4',
... exp='historical',
... ensemble='r1i1p1',
... version='*',
...)
[ESGFFile:cmip5/output1/NCAR/CCSM4/historical/mon/atmos/Amon/r1i1p1/v20121031/tas_
↪Amon_CCSM4_historical_r1i1p1_185001-200512.nc,
ESGFFile:cmip5/output1/NCAR/CCSM4/historical/mon/atmos/Amon/r1i1p1/v20130425/tas_
↪Amon_CCSM4_historical_r1i1p1_185001-200512.nc,
ESGFFile:cmip5/output1/NCAR/CCSM4/historical/mon/atmos/Amon/r1i1p1/v20160829/tas_
↪Amon_CCSM4_historical_r1i1p1_185001-200512.nc]

```

Search for a specific version of a file:

```

>>> find_files(
... project='CMIP5',
... mip='Amon',
... short_name='tas',
... dataset='CCSM4',
... exp='historical',
... ensemble='r1i1p1',
... version='v20130425',
...)
[ESGFFile:cmip5/output1/NCAR/CCSM4/historical/mon/atmos/Amon/r1i1p1/v20130425/tas_
↪Amon_CCSM4_historical_r1i1p1_185001-200512.nc]

```

**Returns**

A list of files that have been found.

**Return type**

list of ESGFFile

## 9.4.2 esmvalcore.esgf.facets

Find files on the ESGF and download them.

Deprecated since version 2.14.0: This module has been moved to *esmvalcore.io.esgf.facets*. Importing it as *esmvalcore.io.esgf.facets* is deprecated and will be removed in version 2.16.0.

**Functions:**

|                                   |                                                                    |
|-----------------------------------|--------------------------------------------------------------------|
| <code>create_dataset_map()</code> | Create the DATASET_MAP from recipe datasets to ESGF dataset names. |
|-----------------------------------|--------------------------------------------------------------------|

`esmvalcore.esgf.facets.create_dataset_map()`

Create the DATASET\_MAP from recipe datasets to ESGF dataset names.

Run `python -m esmvalcore.io.esgf.facets` to print an up to date map.

## 9.5 Exceptions

Exceptions that may be raised by ESMValCore.

**Exceptions:**

|                                            |                                                                |
|--------------------------------------------|----------------------------------------------------------------|
| <code>ESMValCoreDeprecationWarning</code>  | Custom deprecation warning.                                    |
| <code>ESMValCoreLoadWarning</code>         | Custom load warning.                                           |
| <code>ESMValCorePreprocessorWarning</code> | Custom preprocessor warning.                                   |
| <code>ESMValCoreUserWarning</code>         | Base class from which other warnings are derived.              |
| <code>Error</code>                         | Base class from which other exceptions are derived.            |
| <code>InputFilesNotFound(msg)</code>       | Files that are required to run the recipe have not been found. |
| <code>InvalidConfigParameter</code>        | Config parameter is invalid.                                   |
| <code>MissingConfigParameter</code>        | Config parameter is missing.                                   |
| <code>RecipeError(msg)</code>              | Recipe contains an error.                                      |

continues on next page

Table 47 – continued from previous page

|                        |                                                                        |
|------------------------|------------------------------------------------------------------------|
| <i>SuppressedError</i> | Errors subclassed from <i>SuppressedError</i> hide the full traceback. |
|------------------------|------------------------------------------------------------------------|

**exception** `esmvalcore.exceptions.ESMValCoreDeprecationWarning`  
 Bases: *ESMValCoreUserWarning*  
 Custom deprecation warning.

**exception** `esmvalcore.exceptions.ESMValCoreLoadWarning`  
 Bases: *ESMValCorePreprocessorWarning*  
 Custom load warning.

**exception** `esmvalcore.exceptions.ESMValCorePreprocessorWarning`  
 Bases: *ESMValCoreUserWarning*  
 Custom preprocessor warning.

**exception** `esmvalcore.exceptions.ESMValCoreUserWarning`  
 Bases: *UserWarning*  
 Base class from which other warnings are derived.

**exception** `esmvalcore.exceptions.Error`  
 Bases: *Exception*  
 Base class from which other exceptions are derived.

**exception** `esmvalcore.exceptions.InputFilesNotFound(msg: str)`  
 Bases: *RecipeError*  
 Files that are required to run the recipe have not been found.

**Parameters**  
**msg** (*str*)

**Return type**  
 None

**exception** `esmvalcore.exceptions.InvalidConfigParameter`  
 Bases: *Error*, *SuppressedError*  
 Config parameter is invalid.

**exception** `esmvalcore.exceptions.MissingConfigParameter`  
 Bases: *ESMValCoreUserWarning*  
 Config parameter is missing.

**exception** `esmvalcore.exceptions.RecipeError(msg: str)`  
 Bases: *Error*, *SuppressedError*  
 Recipe contains an error.

**Parameters**  
**msg** (*str*)

**Return type**  
 None

**exception** `esmvalcore.exceptions.SuppresseError`Bases: `Exception`Errors subclassed from `SuppresseError` hide the full traceback.

This can be used for simple user-facing errors that do not need the full traceback.

## 9.6 Access data from any source

ESMValCore supports a modular system for reading data from various data sources. In the future, this module may be extended with support for writing output data.

The interface is defined in the `esmvalcore.io.protocol` module and the other modules here provide an implementation for a particular data source.

### 9.6.1 `esmvalcore.io`

A modular system for reading input data from various sources.

An input data source can be defined in the configuration by using `esmvalcore.config.CFG`, for example:

```
>>> from esmvalcore.config import CFG
>>> CFG["projects"]["CMIP6"]["data"]["local"] = {
 "type": "esmvalcore.io.local.LocalDataSource",
 "rootpath": "~/climate_data",
 "dirname_template": "{project}/{activity}/{institute}/{dataset}/{exp}/{ensemble}/
↪{mip}/{short_name}/{grid}/{version}",
 "filename_template": "{short_name}_{mip}_{dataset}_{exp}_{ensemble}_{grid}*.nc",
 }

```

or as a *YAML configuration file*:

```
projects:
 CMIP6:
 data:
 local:
 type: "esmvalcore.io.local.LocalDataSource"
 rootpath: "~/climate_data"
 dirname_template: "{project}/{activity}/{institute}/{dataset}/{exp}/{ensemble}/
↪{mip}/{short_name}/{grid}/{version}"
 filename_template: "{short_name}_{mip}_{dataset}_{exp}_{ensemble}_{grid}*.nc"

```

where `CMIP6` is a project, and `local` is a unique name describing the data source. The data source type, `esmvalcore.io.local.LocalDataSource`, in the example above, needs to implement the `esmvalcore.io.protocol.DataSource` protocol. Any remaining key-value pairs in the configuration, `rootpath`, `dirname_template`, and `filename_template` in this example, are passed as keyword arguments to the data source when it is created.

If there are multiple data sources configured for a project, deduplication of search results happens based on the `esmvalcore.io.protocol.DataElement.name` attribute and the "version" facet in `esmvalcore.io.protocol.DataElement.facets` of the data elements provided by the data sources. If no version facet is specified in the search, the latest version will be used. If there is a tie, the data element provided by the data source with the lowest value of `esmvalcore.io.protocol.DataSource.priority` is chosen.

**Functions:**

|                                                    |                                         |
|----------------------------------------------------|-----------------------------------------|
| <code>load_data_sources(session[, project])</code> | Get the list of available data sources. |
|----------------------------------------------------|-----------------------------------------|

```
esmvalcore.io.load_data_sources(session: Session, project: str | None = None) →
 list[esmvalcore.io.protocol.DataSource]
```

Get the list of available data sources.

If no priority is configured for a data source, the default priority of 1 is used.

#### Parameters

- **session** (*Session*) – The configuration.
- **project** (*str* / *None*) – If specified, only data sources for this project are returned.

#### Returns

A list of available data sources.

#### Raises

**ValueError:** – If the project or its settings are not found in the configuration.

#### Return type

`list[esmvalcore.io.protocol.DataSource]`

## Submodules

### esmvalcore.io.esgf

Find files on the ESGF and download them.

#### Note

This module uses `esgf-pyclient` to search for and download files from the Earth System Grid Federation (ESGF). `esgf-pyclient` uses a [deprecated API](#) that is scheduled to be taken offline and replaced by new APIs based on STAC (ESGF East) and Globus (ESGF West). An ESGF node mimicking the deprecated API but built on top of Globus will be kept online for some time at <https://esgf-node.ornl.gov/esgf-1-5-bridge>, but users are encouraged to migrate to the new APIs as soon as possible by using the `esmvalcore.io.intake_esgf` module instead.

This module provides the function `esmvalcore.io.esgf.find_files()` for searching for files on ESGF using the ESMValTool vocabulary. It returns `esmvalcore.io.esgf.ESGFFile` objects, which have a convenient `esmvalcore.io.esgf.ESGFFile.download()` method for downloading the file. A `esmvalcore.io.esgf.download()` function for downloading multiple files in parallel is also available.

It also provides an `esmvalcore.io.esgf.ESGFDataSource` that can be used to find files on ESGF from the `Dataset` or the `recipe`. To use it, run the command

```
esmvalcore config copy data-esmvalcore-esgf.yml
```

to copy the default configuration file for this module to your configuration directory. This will create a file with the following content:

Listing 1: Contents of `data-esmvalcore-esgf.yml`

```
Download CMIP, CORDEX, and obs4MIPs data from ESGF using the `esmvalcore.io.esgf`
module, which uses the legacy ESGF search interface.
projects:
 CMIP6: &esgf-pyclient-data
```

(continues on next page)

(continued from previous page)

```

data:
 esgf-pyclient:
 type: esmvalcore.io.esgf.ESGFDataSource
 download_dir: ~/climate_data
 # Use a lower priority than for esmvalcore.io.local.LocalDataSource
 # to avoid searching ESGF with the setting `search_esgf: when_missing`.
 priority: 10
CMIP5:
 <<: *esgf-pyclient-data
CMIP3:
 <<: *esgf-pyclient-data
CORDEX:
 <<: *esgf-pyclient-data
obs4MIPs:
 <<: *esgf-pyclient-data

```

See *Data sources* for more information on configuring data sources and *ESGF configuration* for additional configuration options of this module.

**Classes:**

|                                                           |                   |
|-----------------------------------------------------------|-------------------|
| <code>ESGFDataSource(name, project, priority, ...)</code> |                   |
| <code>ESGFFile(results[, dest_folder])</code>             | File on the ESGF. |

**Functions:**

|                                                               |                                          |
|---------------------------------------------------------------|------------------------------------------|
| <code>download(files[, dest_folder, n_jobs])</code>           | Download multiple ESGFFiles in parallel. |
| <code>find_files(*, project, short_name, dataset, ...)</code> | Search for files on ESGF.                |

**class** `esmvalcore.io.esgf.ESGFDataSource`(*name*: 'str', *project*: 'str', *priority*: 'int', *download\_dir*: 'Path')

Bases: *DataSource*

**Attributes:**

|                           |                                                              |
|---------------------------|--------------------------------------------------------------|
| <code>debug_info</code>   | A string containing debug information when no data is found. |
| <code>download_dir</code> | The destination directory where data will be downloaded.     |
| <code>name</code>         | A name identifying the data source.                          |
| <code>priority</code>     | The priority of the data source.                             |
| <code>project</code>      | The project that the data source provides data for.          |

**Methods:**

|                                  |            |
|----------------------------------|------------|
| <code>find_data(**facets)</code> | Find data. |
|----------------------------------|------------|

**Parameters**

- `name` (*str*)

- **project** (*str*)
- **priority** (*int*)
- **download\_dir** (*Path*)

**debug\_info:** *str* = ''

A string containing debug information when no data is found.

**download\_dir:** *Path*

The destination directory where data will be downloaded.

**find\_data**(*\*\*facets: FacetValue*) → *list[ESGFFile]*

Find data.

**Parameters**

**\*\*facets** (*FacetValue*) – Find data matching these facets.

**Returns**

A list of files that have been found on ESGF.

**Return type**

*list* of *esmvalcore.io.esgf.ESGFFile*

**name:** *str*

A name identifying the data source.

**priority:** *int*

The priority of the data source. Lower values have priority.

**project:** *str*

The project that the data source provides data for.

**class** *esmvalcore.io.esgf.ESGFFile*(*results: Iterable[FileResult]*, *dest\_folder: Path | None = None*)

Bases: *DataElement*

File on the ESGF.

This is the object returned by *esmvalcore.io.esgf.find\_files()*.

**Parameters**

- **results** (*Iterable[FileResult]*)
- **dest\_folder** (*Path | None*)

**dataset**

The name of the dataset that the file is part of.

**Type**

*str*

**facets**

Facets describing the file.

**Type**

*dict[str, str]*

**name**

The name of the file.

**Type**

*str*

**size**

The size of the file in bytes.

**Type**

int

**urls**

The URLs where the file can be downloaded.

**Type**

list[str]

**Attributes:**

|                   |                                                     |
|-------------------|-----------------------------------------------------|
| <i>attributes</i> | Attributes are key-value pairs describing the data. |
|-------------------|-----------------------------------------------------|

**Methods:**

|                                 |                                                   |
|---------------------------------|---------------------------------------------------|
| <i>download</i> (dest_folder)   | Download the file.                                |
| <i>local_file</i> (dest_folder) | Return the path to the local file after download. |
| <i>prepare</i> ()               | Prepare the data for access.                      |
| <i>to_iris</i> ()               | Load the data as Iris cubes.                      |

**property attributes:** dict[str, Any]

Attributes are key-value pairs describing the data.

**download**(dest\_folder: Path | None) → LocalFile

Download the file.

**Parameters**

**dest\_folder** (Path | None) – The destination folder.

**Raises**

**DownloadError:** – Raised if downloading the file failed.

**Returns**

The path where the file will be located after download.

**Return type**

LocalFile

**local\_file**(dest\_folder: Path | None) → LocalFile

Return the path to the local file after download.

**Parameters**

**dest\_folder** (Path | None) – The destination folder.

**Returns**

The path where the file will be located after download.

**Return type**

LocalFile

**prepare**() → None

Prepare the data for access.

**Return type**

None

`to_iris()` → `iris.cube.CubeList`

Load the data as Iris cubes.

**Returns**

The loaded data.

**Return type**

`iris.cube.CubeList`

`esmvalcore.io.esgf.download(files, dest_folder=None, n_jobs=4)`

Download multiple ESGFFiles in parallel.

**Parameters**

- **files** (list of *ESGFFile*) – The files to download.
- **dest\_folder** (*Path* or *None*) – The destination folder.
- **n\_jobs** (*int*) – The number of files to download in parallel.

**Raises**

**DownloadError**: – Raised if one or more files failed to download.

`esmvalcore.io.esgf.find_files(*, project, short_name, dataset, **facets)`

Search for files on ESGF.

**Parameters**

- **project** (*str*) – Choose from CMIP3, CMIP5, CMIP6, CORDEX, or obs4MIPs.
- **short\_name** (*str*) – The name of the variable.
- **dataset** (*str*) – The name of the dataset.
- **\*\*facets** (*Union[str, list[str]]*) – Any other search facets. An '\*' can be used to match any value. By default, only the latest version of a file will be returned. To select all versions use `version='*'` while other omitted facets will default to '\*'. It is also possible to specify multiple values for a facet, e.g. `exp=['historical', 'ssp585']` will match any file that belongs to either the historical or ssp585 experiment. The `timerange` facet can be specified in *ISO 8601* format.

**Note**

A value of `timerange='*'` is supported, but combining a '\*' with a time or period *as supported in the recipe* is currently not supported and will return all found files.

**Examples**

Examples of how to use this function for all supported projects.

Search for a CMIP3 dataset:

```
>>> find_files(
... project='CMIP3',
... frequency='mon',
... short_name='tas',
... dataset='cccma_cgcm3_1',
... exp='historical',
... ensemble='run1',
```

(continues on next page)

(continued from previous page)

```
...)
[ESGFFile:cmip3/CCCma/cccma_cgcm3_1/historical/mon/atmos/run1/tas/v1/tas_a1_20c3m_1_
↪cgcm3.1_t47_1850_2000.nc]
```

Search for a CMIP5 dataset:

```
>>> find_files(
... project='CMIP5',
... mip='Amon',
... short_name='tas',
... dataset='inmcm4',
... exp='historical',
... ensemble='r1i1p1',
...)
[ESGFFile:cmip5/output1/INM/inmcm4/historical/mon/atmos/Amon/r1i1p1/v20130207/tas_
↪Amon_inmcm4_historical_r1i1p1_185001-200512.nc]
```

Search for a CMIP6 dataset:

```
>>> find_files(
... project='CMIP6',
... mip='Amon',
... short_name='tas',
... dataset='CanESM5',
... exp='historical',
... ensemble='r1i1p1f1',
...)
[ESGFFile:CMIP6/CMIP/CCCma/CanESM5/historical/r1i1p1f1/Amon/tas/gn/v20190429/tas_
↪Amon_CanESM5_historical_r1i1p1f1_gn_185001-201412.nc]
```

Search for a CORDEX dataset and limit the search results to files containing data to the years in the range 1990-2000:

```
>>> find_files(
... project='CORDEX',
... frequency='mon',
... dataset='COSMO-crCLIM-v1-1',
... short_name='tas',
... exp='historical',
... ensemble='r1i1p1',
... domain='EUR-11',
... driver='MPI-M-MPI-ESM-LR',
... timerange='1990/2000',
...)
[ESGFFile:cordex/output/EUR-11/CLMcom-ETH/MPI-M-MPI-ESM-LR/historical/r1i1p1/COSMO-
↪crCLIM-v1-1/v1/mon/tas/v20191219/tas_EUR-11_MPI-M-MPI-ESM-LR_historical_r1i1p1_
↪CLMcom-ETH-COSMO-crCLIM-v1-1_v1_mon_198101-199012.nc,
ESGFFile:cordex/output/EUR-11/CLMcom-ETH/MPI-M-MPI-ESM-LR/historical/r1i1p1/COSMO-
↪crCLIM-v1-1/v1/mon/tas/v20191219/tas_EUR-11_MPI-M-MPI-ESM-LR_historical_r1i1p1_
↪CLMcom-ETH-COSMO-crCLIM-v1-1_v1_mon_199101-200012.nc]
```

Search for an obs4MIPs dataset:

```
>>> find_files(
... project='obs4MIPs',
... frequency='mon',
... dataset='CERES-EBAF',
... short_name='rsutcs',
...)
[ESGFFile:obs4MIPs/NASA-LaRC/CERES-EBAF/atmos/mon/v20160610/rsutcs_CERES-EBAF_L3B_
↪Ed2-8_200003-201404.nc]
```

Search for any ensemble member:

```
>>> find_files(
... project='CMIP6',
... mip='Amon',
... short_name='tas',
... dataset='BCC-CSM2-MR',
... exp='historical',
... ensemble='*',
...)
[ESGFFile:CMIP6/CMIP/BCC/BCC-CSM2-MR/historical/r1i1p1f1/Amon/tas/gn/v20181126/tas_
↪Amon_BCC-CSM2-MR_historical_r1i1p1f1_gn_185001-201412.nc,
ESGFFile:CMIP6/CMIP/BCC/BCC-CSM2-MR/historical/r2i1p1f1/Amon/tas/gn/v20181115/tas_
↪Amon_BCC-CSM2-MR_historical_r2i1p1f1_gn_185001-201412.nc,
ESGFFile:CMIP6/CMIP/BCC/BCC-CSM2-MR/historical/r3i1p1f1/Amon/tas/gn/v20181119/tas_
↪Amon_BCC-CSM2-MR_historical_r3i1p1f1_gn_185001-201412.nc]
```

Search for all available versions of a file:

```
>>> find_files(
... project='CMIP5',
... mip='Amon',
... short_name='tas',
... dataset='CCSM4',
... exp='historical',
... ensemble='r1i1p1',
... version='*',
...)
[ESGFFile:cmip5/output1/NCAR/CCSM4/historical/mon/atmos/Amon/r1i1p1/v20121031/tas_
↪Amon_CCSM4_historical_r1i1p1_185001-200512.nc,
ESGFFile:cmip5/output1/NCAR/CCSM4/historical/mon/atmos/Amon/r1i1p1/v20130425/tas_
↪Amon_CCSM4_historical_r1i1p1_185001-200512.nc,
ESGFFile:cmip5/output1/NCAR/CCSM4/historical/mon/atmos/Amon/r1i1p1/v20160829/tas_
↪Amon_CCSM4_historical_r1i1p1_185001-200512.nc]
```

Search for a specific version of a file:

```
>>> find_files(
... project='CMIP5',
... mip='Amon',
... short_name='tas',
... dataset='CCSM4',
... exp='historical',
... ensemble='r1i1p1',
```

(continues on next page)

(continued from previous page)

```

... version='v20130425',
...)
[ESGFFile:cmip5/output1/NCAR/CCSM4/historical/mon/atmos/Amon/r1i1p1/v20130425/tas_
↪Amon_CCSM4_historical_r1i1p1_185001-200512.nc]

```

**Returns**

A list of files that have been found.

**Return type**

list of *ESGFFile*

**esmvalcore.io.esgf.facets**

Module containing mappings from our names to ESGF names.

Note that only a subset of the available facets is used for searching, to minimize issues due to wrong facet values on ESGF. A complete mapping from ESMValCore facet names to the names used on ESGF is available in *Facets*.

**Data:**

|                    |                                                                         |
|--------------------|-------------------------------------------------------------------------|
| <i>DATASET_MAP</i> | Cache for the mapping between recipe/filesystem and ESGF dataset names. |
| <i>FACETS</i>      | Mapping between the recipe and ESGF facet names.                        |

**Functions:**

|                             |                                                                    |
|-----------------------------|--------------------------------------------------------------------|
| <i>create_dataset_map()</i> | Create the DATASET_MAP from recipe datasets to ESGF dataset names. |
|-----------------------------|--------------------------------------------------------------------|

```

esmvalcore.io.esgf.facets.DATASET_MAP = {'CMIP3': {}, 'CMIP5': {'ACCESS1-0':
'ACCESS1.0', 'ACCESS1-3': 'ACCESS1.3', 'CESM1-BGC': 'CESM1(BGC)', 'CESM1-CAM5':
'CESM1(CAM5)', 'CESM1-CAM5-1-FV2': 'CESM1(CAM5.1,FV2)', 'CESM1-FASTCHEM':
'CESM1(FASTCHEM)', 'CESM1-WACCM': 'CESM1(WACCM)', 'CSIRO-Mk3-6-0': 'CSIRO-Mk3.6.0',
'GFDL-CM2p1': 'GFDL-CM2.1', 'MRI-AGCM3-2H': 'MRI-AGCM3.2H', 'MRI-AGCM3-2S':
'MRI-AGCM3.2S', 'bcc-csm1-1': 'BCC-CSM1.1', 'bcc-csm1-1-m': 'BCC-CSM1.1(m)', 'fio-esm':
'FIO-ESM', 'inmcm4': 'INM-CM4'}, 'CMIP6': {}, 'CORDEX': {}, 'obs4MIPs': {}}

```

Cache for the mapping between recipe/filesystem and ESGF dataset names.

```
esmvalcore.io.esgf.facets.FACETS = {'CMIP3': {'dataset': 'model', 'ensemble':
'ensemble', 'exp': 'experiment', 'frequency': 'time_frequency', 'short_name':
'variable'}, 'CMIP5': {'dataset': 'model', 'ensemble': 'ensemble', 'exp':
'experiment', 'frequency': 'time_frequency', 'institute': 'institute', 'mip':
'cmor_table', 'product': 'product', 'short_name': 'variable'}, 'CMIP6': {'activity':
'activity_drs', 'dataset': 'source_id', 'ensemble': 'member_id', 'exp':
'experiment_id', 'grid': 'grid_label', 'institute': 'institution_id', 'mip':
'table_id', 'short_name': 'variable'}, 'CMIP7': {'activity': 'activity_id',
'branding_suffix': 'branding_suffix', 'dataset': 'source_id', 'ensemble':
'variant_label', 'exp': 'experiment_id', 'frequency': 'frequency', 'grid':
'grid_label', 'institute': 'institution_id', 'mip': 'realm', 'region': 'region',
'short_name': 'variable_id'}, 'CORDEX': {'dataset': 'rcm_name', 'domain': 'domain',
'driver': 'driving_model', 'ensemble': 'ensemble', 'exp': 'experiment', 'frequency':
'time_frequency', 'institute': 'institute', 'product': 'product', 'short_name':
'variable'}, 'obs4MIPs': {'dataset': 'source_id', 'frequency': 'time_frequency',
'institute': 'institute', 'short_name': 'variable'}}
```

Mapping between the recipe and ESGF facet names.

```
esmvalcore.io.esgf.facets.create_dataset_map()
```

Create the DATASET\_MAP from recipe datasets to ESGF dataset names.

Run `python -m esmvalcore.io.esgf.facets` to print an up to date map.

## esmvalcore.io.intake\_esgf

Access data using `intake-esgf`.

### Note

It is highly recommended that you take a moment to [configure intake-esgf](#) before using it with ESMValCore. Make sure to set `local_cache` to a path where it can store downloaded files and if (some) ESGF data is already available on your system, point `esg_dataroot` to it. If you are missing certain search results, you may want to choose a different index node for searching the ESGF.

Run the command `esmvaltool config copy data-intake-esgf.yml` to update your *configuration* to use this module. This will create a file with the following content in your configuration directory:

Listing 2: Contents of `data-intake-esgf.yml`

```
Read CMIP and obs4MIPs data from ESGF using intake-esgf.
projects:
 CMIP6:
 data:
 intake-esgf:
 type: esmvalcore.io.intake_esgf.IntakeESGFDataSource
 facets:
 activity: "activity_drs"
 dataset: "source_id"
 ensemble: "member_id"
 exp: "experiment_id"
 institute: "institution_id"
 grid: "grid_label"
 mip: "table_id"
```

(continues on next page)

(continued from previous page)

```

 project: "project"
 short_name: "variable_id"
CMIP5:
 data:
 intake-esgf:
 type: esmvalcore.io.intake_esgf.IntakeESGFDataSource
 facets:
 dataset: "model"
 ensemble: "ensemble"
 exp: "experiment"
 frequency: "time_frequency"
 institute: "institute"
 mip: "cmor_table"
 product: "product"
 project: "project"
 short_name: "variable"
 values:
 dataset:
 "ACCESS1-0": "ACCESS1.0"
 "ACCESS1-3": "ACCESS1.3"
 "bcc-csm1-1": "BCC-CSM1.1"
 "bcc-csm1-1-m": "BCC-CSM1.1(m)"
 "CESM1-BGC": "CESM1(BGC)"
 "CESM1-CAM5": "CESM1(CAM5)"
 "CESM1-CAM5-1-FV2": "CESM1(CAM5.1,FV2)"
 "CESM1-FASTCHEM": "CESM1(FASTCHEM)"
 "CESM1-WACCM": "CESM1(WACCM)"
 "CSIRO-Mk3-6-0": "CSIRO-Mk3.6.0"
 "fio-esm": "FIO-ESM"
 "GFDL-CM2p1": "GFDL-CM2.1"
 "inmcm4": "INM-CM4"
 "MRI-AGCM3-2H": "MRI-AGCM3.2H"
 "MRI-AGCM3-2S": "MRI-AGCM3.2S"
CMIP3:
 data:
 intake-esgf:
 type: esmvalcore.io.intake_esgf.IntakeESGFDataSource
 facets:
 dataset: "model"
 ensemble: "ensemble"
 exp: "experiment"
 frequency: "time_frequency"
 project: "project"
 short_name: "variable"
obs4MIPs:
 data:
 intake-esgf-v2:
 type: esmvalcore.io.intake_esgf.IntakeESGFDataSource
 facets:
 dataset: "source_id"
 frequency: "frequency"
 institute: "institution_id"

```

(continues on next page)

(continued from previous page)

```

project: "project"
short_name: "variable_id"
TODO: Add support for older ODS V1.0 obs4MIPs (CMIP5 style) data to intake-esgf
intake-esgf-v1:
type: esmvalcore.io.intake_esgf.IntakeESGFDataSource
facets:
dataset: "source_id"
frequency: "time_frequency"
institute: "institute"
project: "project"
short_name: "variable"

```

**Classes:**

|                                                        |                                                                  |
|--------------------------------------------------------|------------------------------------------------------------------|
| <code>IntakeESGFDataSource</code> (name, project, ...) | Data source that can be used to find data using intake-esgf.     |
| <code>IntakeESGFDataset</code> (name, facets, catalog) | A dataset that can be used to load data found using intake-esgf. |

```

class esmvalcore.io.intake_esgf.IntakeESGFDataSource(name: str, project: str, priority: int, facets:
 dict[str, str], values: dict[str, dict[str, str]] =
 <factory>)

```

Bases: `DataSource`

Data source that can be used to find data using intake-esgf.

**Attributes:**

|                         |                                                              |
|-------------------------|--------------------------------------------------------------|
| <code>catalog</code>    | The intake-esgf catalog used to find data.                   |
| <code>debug_info</code> | A string containing debug information when no data is found. |
| <code>facets</code>     | Mapping between the ESMValCore and ESGF facet names.         |
| <code>name</code>       | A name identifying the data source.                          |
| <code>priority</code>   | The priority of the data source.                             |
| <code>project</code>    | The project that the data source provides data for.          |
| <code>values</code>     | Mapping between the ESMValCore and ESGF facet values.        |

**Methods:**

|                                   |            |
|-----------------------------------|------------|
| <code>find_data</code> (**facets) | Find data. |
|-----------------------------------|------------|

**Parameters**

- **name** (*str*)
- **project** (*str*)
- **priority** (*int*)
- **facets** (*dict[str, str]*)

- **values** (*dict[str, dict[str, str]]*)

**catalog:** `ESGFCatalog`

The intake-esgf catalog used to find data.

**debug\_info:** `str = ''`

A string containing debug information when no data is found.

**facets:** `dict[str, str]`

Mapping between the ESMValCore and ESGF facet names.

**find\_data**(*\*\*facets: FacetValue*) → *list[IntakeESGFDataset]*

Find data.

**Parameters**

**\*\*facets** (*FacetValue*) – Find data matching these facets.

**Returns**

A list of data elements that have been found.

**Return type**

*list[IntakeESGFDataset]*

**name:** `str`

A name identifying the data source.

**priority:** `int`

The priority of the data source. Lower values have priority.

**project:** `str`

The project that the data source provides data for.

**values:** `dict[str, dict[str, str]]`

Mapping between the ESMValCore and ESGF facet values.

**class** `esmvalcore.io.intake_esgf.IntakeESGFDataset`(*name: str, facets: Facets, catalog: intake\_esgf.ESGFCatalog*)

Bases: *DataElement*

A dataset that can be used to load data found using `intake-esgf`.

**Attributes:**

|                   |                                                              |
|-------------------|--------------------------------------------------------------|
| <i>attributes</i> | Attributes are key-value pairs describing the data.          |
| <i>catalog</i>    | The intake-esgf catalog describing this data.                |
| <i>facets</i>     | Facets are key-value pairs that were used to find this data. |
| <i>name</i>       | A unique name identifying the data.                          |

**Methods:**

|                  |                              |
|------------------|------------------------------|
| <i>prepare()</i> | Prepare the data for access. |
| <i>to_iris()</i> | Load the data as Iris cubes. |

**Parameters**

- **name** (*str*)

- **facets** (*Facets*)
- **catalog** (*intake\_esgf.ESGFCatalog*)

**property attributes:** `dict[str, Any]`

Attributes are key-value pairs describing the data.

**catalog:** `intake_esgf.ESGFCatalog`

The intake-esgf catalog describing this data.

**facets:** `Facets`

Facets are key-value pairs that were used to find this data.

**name:** `str`

A unique name identifying the data.

**prepare()** → `None`

Prepare the data for access.

**Return type**

`None`

**to\_iris()** → `iris.cube.CubeList`

Load the data as Iris cubes.

**Returns**

The loaded data.

**Return type**

`iris.cube.CubeList`

## esmvalcore.io.local

Find files on the local filesystem.

Example configuration to find CMIP6 data on a personal computer:

Listing 3: Contents of `data-local.yml`

```

projects:
 CMIP7:
 data:
 local:
 type: esmvalcore.io.local.LocalDataSource
 rootpath: ~/climate_data
 # Directory structure and file name template from https://wcrp-cmip.github.io/
 ↪ cmip7-guidance/CMIP7/global_attributes/
 dirname_template: "{drs_specs}/{project}/{activity}/{institute}/{dataset}/{exp}/
 ↪ {ensemble}/{region}/{frequency}/{short_name}/{branding_suffix}/{grid}/{version}"
 filename_template: "{short_name}_{branding_suffix}_{frequency}_{region}_{grid}_
 ↪ {dataset}_{exp}_{ensemble}*.nc"
 CMIP6:
 data:
 local:
 type: esmvalcore.io.local.LocalDataSource
 rootpath: ~/climate_data
 dirname_template: "{project}/{activity}/{institute}/{dataset}/{exp}/{ensemble}/

```

(continues on next page)

(continued from previous page)

```
↪{mip}/{short_name}/{grid}/{version}"
filename_template: "{short_name}_{mip}_{dataset}_{exp}_{ensemble}_{grid}*.nc"
```

The module will find files matching the `glob.glob()` pattern formed by `rootpath/dirname_template/` `filename_template`, where the facets defined inside the curly braces of the templates are replaced by their values from the *Dataset* or the *recipe* plus any facet-value pairs that can be automatically added using `augment_facets()`. Note that the name of the data source, `local-data` in the example above, must be unique within each project but can otherwise be chosen freely.

To start using this module on a personal computer, copy the example configuration file into your configuration directory by running the command:

```
esmvaltool config copy data-local.yml
```

and tailor it for your own system if needed.

Example configuration files for popular HPC systems and some *supported climate models* are also available. View the list of available files by running the command:

```
esmvaltool config list
```

Further information is available in *Data sources*.

**Data:**

|                     |                       |
|---------------------|-----------------------|
| <i>GRIB_FORMATS</i> | GRIB file extensions. |
|---------------------|-----------------------|

**Classes:**

|                                                       |                                                      |
|-------------------------------------------------------|------------------------------------------------------|
| <i>LocalDataSource</i> (name, project, priority, ...) | Data source for finding files on a local filesystem. |
| <i>LocalFile</i> (*args, **kwargs)                    | File on the local filesystem.                        |

```
esmvalcore.io.local.GRIB_FORMATS = ('.grib2', '.grib', '.grb2', '.grb', '.gb2', '.gb')
GRIB file extensions.
```

```
class esmvalcore.io.local.LocalDataSource(name: str, project: str, priority: int, rootpath: ~pathlib.Path,
dirname_template: str, filename_template: str,
ignore_warnings: list[dict[str, ~typing.Any]] | None =
<factory>)
```

Bases: *DataSource*

Data source for finding files on a local filesystem.

**Attributes:**

|                          |                                                              |
|--------------------------|--------------------------------------------------------------|
| <i>debug_info</i>        | A string containing debug information when no data is found. |
| <i>dirname_template</i>  | The template for the directory names.                        |
| <i>filename_template</i> | The template for the file names.                             |
| <i>ignore_warnings</i>   | Warnings to ignore when loading the data.                    |
| <i>name</i>              | A name identifying the data source.                          |
| <i>priority</i>          | The priority of the data source.                             |

continues on next page

Table 64 – continued from previous page

|                       |                                                     |
|-----------------------|-----------------------------------------------------|
| <code>project</code>  | The project that the data source provides data for. |
| <code>rootpath</code> | The path where the directories are located.         |

**Methods:**

|                                  |                    |
|----------------------------------|--------------------|
| <code>find_data(**facets)</code> | Find data locally. |
|----------------------------------|--------------------|

**Parameters**

- **name** (*str*)
- **project** (*str*)
- **priority** (*int*)
- **rootpath** (*Path*)
- **dirname\_template** (*str*)
- **filename\_template** (*str*)
- **ignore\_warnings** (*list[dict[str, Any]] | None*)

**debug\_info:** `str = ''`

A string containing debug information when no data is found.

**dirname\_template:** `str`

The template for the directory names.

**filename\_template:** `str`

The template for the file names.

**find\_data**(\*\**facets*: *FacetValue*) → *list[LocalFile]*

Find data locally.

**Parameters**

**\*\*facets** (*FacetValue*) – Find data matching these facets.

**Returns**

A list of files.

**Return type**

*list[LocalFile]*

**ignore\_warnings:** `list[dict[str, Any]] | None`

Warnings to ignore when loading the data.

The `list` should contain `dict`s` with keyword arguments that will be passed to the `:func:`warnings.filterwarnings` function when calling `LocalFile.to_iris()`.

**name:** `str`

A name identifying the data source.

**priority:** `int`

The priority of the data source. Lower values have priority.

**project:** `str`

The project that the data source provides data for.

**rootpath:** `Path`

The path where the directories are located.

**class** `esmvalcore.io.local.LocalFile(*args, **kwargs)`

Bases: `PosixPath`, `DataElement`

File on the local filesystem.

**Attributes:**

|                              |                                                              |
|------------------------------|--------------------------------------------------------------|
| <code>attributes</code>      | Attributes are key-value pairs describing the data.          |
| <code>facets</code>          | Facets are key-value pairs that were used to find this data. |
| <code>ignore_warnings</code> | Warnings to ignore when loading the data.                    |

**Methods:**

|                        |                              |
|------------------------|------------------------------|
| <code>prepare()</code> | Prepare the data for access. |
| <code>to_iris()</code> | Load the data as Iris cubes. |

**property attributes:** `dict[str, Any]`

Attributes are key-value pairs describing the data.

**property facets:** `Facets`

Facets are key-value pairs that were used to find this data.

**property ignore\_warnings:** `list[dict[str, Any]] | None`

Warnings to ignore when loading the data.

The list should contain dict's with keyword arguments that will be passed to the `:func:`warnings.filterwarnings`` function when calling the `to_iris` method.

**prepare()** → `None`

Prepare the data for access.

**Return type**

`None`

**to\_iris()** → `CubeList`

Load the data as Iris cubes.

**Returns**

The loaded data.

**Return type**

`iris.cube.CubeList`

## esmvalcore.io.protocol

Protocols for accessing data.

This module defines the `DataSource` and `DataElement` protocols for finding and loading data. A data source can be used to find data elements matching specific facets. A data element represents some data that can be loaded as Iris cubes.

To add support for a new data source, write two classes that implement these protocols and configure the tool to use the newly implemented data source as described in `esmvalcore.io`.

**Classes:**

|                                           |                                                         |
|-------------------------------------------|---------------------------------------------------------|
| <code>DataElement(*args, **kwargs)</code> | A data element represents some data that can be loaded. |
| <code>DataSource(*args, **kwargs)</code>  | A data source can be used to find data.                 |

**class** `esmvalcore.io.protocol.DataElement(*args, **kwargs)`

Bases: `Protocol`

A data element represents some data that can be loaded.

An `esmvalcore.io.local.LocalFile` is an example of a data element.

**Attributes:**

|                         |                                                                     |
|-------------------------|---------------------------------------------------------------------|
| <code>attributes</code> | Attributes are key-value pairs describing the data.                 |
| <code>facets</code>     | Facets are key-value pairs that can be used for searching the data. |
| <code>name</code>       | A unique name identifying the data.                                 |

**Methods:**

|                        |                              |
|------------------------|------------------------------|
| <code>prepare()</code> | Prepare the data for access. |
| <code>to_iris()</code> | Load the data as Iris cubes. |

**attributes:** `dict[str, Any]`

Attributes are key-value pairs describing the data.

**facets:** `dict[str, FacetValue]`

Facets are key-value pairs that can be used for searching the data.

**name:** `str`

A unique name identifying the data.

**prepare()** → `None`

Prepare the data for access.

**Return type**

`None`

**to\_iris()** → `iris.cube.CubeList`

Load the data as Iris cubes.

**Returns**

The loaded data.

**Return type**

`iris.cube.CubeList`

**class** `esmvalcore.io.protocol.DataSource(*args, **kwargs)`

Bases: `Protocol`

A data source can be used to find data.

**Attributes:**

|                         |                                                              |
|-------------------------|--------------------------------------------------------------|
| <code>debug_info</code> | A string containing debug information when no data is found. |
| <code>name</code>       | A name identifying the data source.                          |
| <code>priority</code>   | The priority of the data source.                             |
| <code>project</code>    | The project that the data source provides data for.          |

#### Methods:

|                                  |            |
|----------------------------------|------------|
| <code>find_data(**facets)</code> | Find data. |
|----------------------------------|------------|

#### `debug_info: str`

A string containing debug information when no data is found.

#### `find_data(**facets: FacetValue) → Iterable[DataElement]`

Find data.

##### Parameters

**\*\*facets** (*FacetValue*) – Find data matching these facets.

##### Returns

The data elements that have been found.

##### Return type

`typing.Iterable` of `esmvalcore.io.base.DataElement`

#### `name: str`

A name identifying the data source.

#### `priority: int`

The priority of the data source. Lower values have priority.

#### `project: str`

The project that the data source provides data for.

## 9.7 Iris helper functions

Auxiliary functions for `iris`.

#### Functions:

|                                                            |                                                                  |
|------------------------------------------------------------|------------------------------------------------------------------|
| <code>add_leading_dim_to_cube(cube, dim_coord)</code>      | Add new leading dimension to cube.                               |
| <code>dataset_to_iris(dataset[, filepath, ...])</code>     | Convert dataset to <code>CubeList</code> .                       |
| <code>date2num(date, unit[, dtype])</code>                 | Convert datetime object into numeric value with requested dtype. |
| <code>has_irregular_grid(cube)</code>                      | Check if a cube has an irregular grid.                           |
| <code>has_regular_grid(cube)</code>                        | Check if a cube has a regular grid.                              |
| <code>has_unstructured_grid(cube)</code>                   | Check if a cube has an unstructured grid.                        |
| <code>ignore_iris_vague_metadata_warnings()</code>         | Ignore specific warnings.                                        |
| <code>ignore_warnings_context([warnings_to_ignore])</code> | Ignore warnings (context manager).                               |
| <code>merge_cube_attributes(cubes[, delimiter])</code>     | Merge attributes of all given cubes in-place.                    |
| <code>rechunk_cube(cube, complete_coords[, ...])</code>    | Rechunk cube so that it is not chunked along given dimensions.   |

continues on next page

Table 73 – continued from previous page

|                                              |                                                                                    |
|----------------------------------------------|------------------------------------------------------------------------------------|
| <code>safe_convert_units(cube, units)</code> | Safe unit conversion (change of <code>standard_name</code> not allowed; in-place). |
|----------------------------------------------|------------------------------------------------------------------------------------|

`esmvalcore.iris_helpers.add_leading_dim_to_cube(cube: Cube, dim_coord: DimCoord) → Cube`

Add new leading dimension to cube.

An input cube with shape (x, ..., z) will be transformed to a cube with shape (w, x, ..., z) where w is the length of `dim_coord`. Note that the data is broadcasted to the new shape.

#### Parameters

- **cube** (*Cube*) – Input cube.
- **dim\_coord** (*DimCoord*) – Dimensional coordinate that is used to describe the new leading dimension. Needs to be 1D.

#### Returns

Transformed input cube with new leading dimension.

#### Return type

`iris.cube.Cube`

#### Raises

`iris.exceptions.CoordinateMultiDimError` – `dim_coord` is not 1D.

`esmvalcore.iris_helpers.dataset_to_iris(dataset: xr.Dataset | ncdata.NcData, filepath: str | Path | None = None, ignore_warnings: list[dict[str, Any]] | None = None) → iris.cube.CubeList`

Convert dataset to `CubeList`.

#### Parameters

- **dataset** (*xr.Dataset* | *ncdata.NcData*) – The dataset object to convert.
- **filepath** (*str* | *Path* | *None*) – The path that the dataset was loaded from.
- **ignore\_warnings** (*list[dict[str, Any]]* | *None*) – Keyword arguments passed to `warnings.filterwarnings()` used to ignore warnings during data loading. Each list element corresponds to one call to `warnings.filterwarnings()`.

#### Returns

`CubeList` containing the requested cubes.

#### Return type

`iris.cube.CubeList`

#### Raises

`TypeError` – Invalid type for dataset given.

`esmvalcore.iris_helpers.date2num(date, unit, dtype=<class 'numpy.float64'>)`

Convert datetime object into numeric value with requested dtype.

This is a custom version of `cf_units.Unit.date2num()` that guarantees the correct dtype for the return value.

#### Parameters

- **date** (*datetime.datetime* or *cftime.datetime*)
- **unit** (*cf\_units.Unit*)
- **dtype** (*a numpy dtype*)

**Returns**

The return value of `unit.date2num` with the requested dtype.

**Return type**

`numpy.ndarray` of type *dtype*

`esmvalcore.iris_helpers.has_irregular_grid(cube: Cube) → bool`

Check if a cube has an irregular grid.

“Irregular” refers to a general curvilinear grid with 2D latitude and 2D longitude coordinates with common dimensions.

**Parameters**

**cube** (*Cube*) – Cube to be checked.

**Returns**

True if input cube has an irregular grid, else False.

**Return type**

`bool`

`esmvalcore.iris_helpers.has_regular_grid(cube: Cube) → bool`

Check if a cube has a regular grid.

“Regular” refers to a rectilinear grid with 1D latitude and 1D longitude coordinates orthogonal to each other.

**Parameters**

**cube** (*Cube*) – Cube to be checked.

**Returns**

True if input cube has a regular grid, else False.

**Return type**

`bool`

`esmvalcore.iris_helpers.has_unstructured_grid(cube: Cube) → bool`

Check if a cube has an unstructured grid.

“Unstructured” refers to a grid with 1D latitude and 1D longitude coordinates with common dimensions (i.e., a simple list of points).

**Parameters**

**cube** (*Cube*) – Cube to be checked.

**Returns**

True if input cube has an unstructured grid, else False.

**Return type**

`bool`

`esmvalcore.iris_helpers.ignore_iris_vague_metadata_warnings() → Generator[None]`

Ignore specific warnings.

This can be used as a context manager. See also <https://scitools-iris.readthedocs.io/en/stable/generated/api/iris.warnings.html#iris.warnings.IrisVagueMetadataWarning>.

**Return type**

`Generator[None]`

`esmvalcore.iris_helpers.ignore_warnings_context(warnings_to_ignore: list[dict[str, Any]] | None = None) → Generator[None]`

Ignore warnings (context manager).

**Parameters**

**warnings\_to\_ignore** (*list[dict[str, Any]] | None*) – Additional warnings to ignore (by default, Iris warnings about missing CF-netCDF measure variables and invalid units are ignored).

**Return type**

Generator[None]

`esmvalcore.iris_helpers.merge_cube_attributes(cubes: Sequence[Cube], delimiter: str = ' ') → None`

Merge attributes of all given cubes in-place.

After this operation, the attributes of all given cubes are equal. This is useful for operations that combine cubes, such as `iris.cube.CubeList.merge_cube()` or `iris.cube.CubeList.concatenate_cube()`.

**Note**

This function differs from `iris.util.equalise_attributes()` in this respect that it does not delete attributes that are not identical but rather concatenates them (sorted) using the given delimiter. E.g., the attributes `exp: historical` and `exp: ssp585` end up as `exp: historical ssp585` using the default `delimiter = ' '`.

**Parameters**

- **cubes** (*Sequence[Cube]*) – Input cubes whose attributes will be modified in-place.
- **delimiter** (*str*) – Delimiter that is used to concatenate non-identical attributes.

**Return type**

None

`esmvalcore.iris_helpers.rechunk_cube(cube: Cube, complete_coords: Iterable[Coord | str], remaining_dims: int | Literal['auto'] = 'auto') → Cube`

Rechunk cube so that it is not chunked along given dimensions.

This will rechunk the cube's data, but also all non-dimensional coordinates, cell measures, and ancillary variables that span at least one of the given dimensions.

**Note**

This will only rechunk *dask* arrays. *numpy* arrays are not changed.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **complete\_coords** (*Iterable[Coord | str]*) – (Names of) coordinates along which the output cube should not be chunked.
- **remaining\_dims** (*int | Literal['auto']*) – Chunksize of the remaining dimensions.

**Returns**

Rechunked cube. This will always be a copy of the input cube.

**Return type**

`iris.cube.Cube`

`esmvalcore.iris_helpers.safe_convert_units(cube: Cube, units: str | Unit) → Cube`

Safe unit conversion (change of *standard\_name* not allowed; in-place).

This is a safe version of `esmvalcore.preprocessor.convert_units()` that will raise an error if the input cube's *standard\_name* has been changed.

**Parameters**

- **cube** (*Cube*) – Input cube (modified in place).
- **units** (*str* | *Unit*) – New units.

**Returns**

Converted cube. Just returned for convenience; input cube is modified in place.

**Return type**

`iris.cube.Cube`

**Raises**

- **`iris.exceptions.UnitConversionError`** – Old units are unknown.
- **`ValueError`** – Old units are not convertible to new units or unit conversion required change of *standard\_name*.

## 9.8 Find files on the local filesystem

Find files on the local filesystem.

Deprecated since version 2.14.0: This module has been moved to `esmvalcore.io.local`. Importing it as `esmvalcore.local` is deprecated and will be removed in version 2.16.0.

**Classes:**

|                                                            |                                                      |
|------------------------------------------------------------|------------------------------------------------------|
| <code>DataSource(*args, **kwargs)</code>                   | Data source for finding files on a local filesystem. |
| <code>LocalDataSource(name, project, priority, ...)</code> | Data source for finding files on a local filesystem. |
| <code>LocalFile(*args, **kwargs)</code>                    | File on the local filesystem.                        |

**Functions:**

|                                     |                                     |
|-------------------------------------|-------------------------------------|
| <code>find_files(*[, debug])</code> | Find files on the local filesystem. |
|-------------------------------------|-------------------------------------|

**class** `esmvalcore.local.DataSource(*args, **kwargs)`

Bases: `LocalDataSource`

Data source for finding files on a local filesystem.

Deprecated since version 2.14.0: This class is deprecated and will be removed in version 2.16.0. Please use `esmvalcore.local.LocalDataSource` instead.

**Methods:**

|                                               |                                                        |
|-----------------------------------------------|--------------------------------------------------------|
| <code>find_files(**facets)</code>             | Find files.                                            |
| <code>get_glob_patterns(**facets)</code>      | Compose the globs that will be used to look for files. |
| <code>path2facets(path, add_timerange)</code> | Extract facets from path.                              |

**Attributes:**

|                            |                                                                  |
|----------------------------|------------------------------------------------------------------|
| <code>regex_pattern</code> | Get regex pattern that can be used to extract facets from paths. |
|----------------------------|------------------------------------------------------------------|

**find\_files**(*\*\*facets: FacetValue*) → list[LocalFile]

Find files.

**Parameters**

**facets** (*FacetValue*)

**Return type**

list[LocalFile]

**get\_glob\_patterns**(*\*\*facets: FacetValue*) → list[Path]

Compose the globs that will be used to look for files.

**Parameters**

**facets** (*FacetValue*)

**Return type**

list[Path]

**path2facets**(*path: Path, add\_timerange: bool*) → dict[str, str]

Extract facets from path.

**Parameters**

- **path** (*Path*)
- **add\_timerange** (*bool*)

**Return type**

dict[str, str]

**property regex\_pattern: str**

Get regex pattern that can be used to extract facets from paths.

**class** esmvalcore.local.LocalDataSource(*name: str, project: str, priority: int, rootpath: ~pathlib.Path, dirname\_template: str, filename\_template: str, ignore\_warnings: list[dict[str, ~typing.Any]] | None = <factory>*)

Bases: DataSource

Data source for finding files on a local filesystem.

**Attributes:**

|                                |                                                              |
|--------------------------------|--------------------------------------------------------------|
| <code>debug_info</code>        | A string containing debug information when no data is found. |
| <code>dirname_template</code>  | The template for the directory names.                        |
| <code>filename_template</code> | The template for the file names.                             |
| <code>ignore_warnings</code>   | Warnings to ignore when loading the data.                    |
| <code>name</code>              | A name identifying the data source.                          |
| <code>priority</code>          | The priority of the data source.                             |
| <code>project</code>           | The project that the data source provides data for.          |
| <code>rootpath</code>          | The path where the directories are located.                  |

**Methods:**

**`find_data(**facets)`**

Find data locally.

**Parameters**

- **name** (*str*)
- **project** (*str*)
- **priority** (*int*)
- **rootpath** (*Path*)
- **dirname\_template** (*str*)
- **filename\_template** (*str*)
- **ignore\_warnings** (*list[dict[str, Any]]* | *None*)

**debug\_info:** *str* = ''

A string containing debug information when no data is found.

**dirname\_template:** *str*

The template for the directory names.

**filename\_template:** *str*

The template for the file names.

**find\_data(\*\*facets: FacetValue)** → *list[LocalFile]*

Find data locally.

**Parameters****\*\*facets** (*FacetValue*) – Find data matching these facets.**Returns**

A list of files.

**Return type***list[LocalFile]***ignore\_warnings:** *list[dict[str, Any]]* | *None*

Warnings to ignore when loading the data.

The list should contain dict`s with keyword arguments that will be passed to the `:func:`warnings.filterwarnings` function when calling `LocalFile.to_iris()`.**name:** *str*

A name identifying the data source.

**priority:** *int*

The priority of the data source. Lower values have priority.

**project:** *str*

The project that the data source provides data for.

**rootpath:** *Path*

The path where the directories are located.

```
class esmvalcore.local.LocalFile(*args, **kwargs)
```

Bases: `PosixPath`, `DataElement`

File on the local filesystem.

**Attributes:**

|                              |                                                              |
|------------------------------|--------------------------------------------------------------|
| <code>attributes</code>      | Attributes are key-value pairs describing the data.          |
| <code>facets</code>          | Facets are key-value pairs that were used to find this data. |
| <code>ignore_warnings</code> | Warnings to ignore when loading the data.                    |

**Methods:**

|                        |                              |
|------------------------|------------------------------|
| <code>prepare()</code> | Prepare the data for access. |
| <code>to_iris()</code> | Load the data as Iris cubes. |

**property attributes:** `dict[str, Any]`

Attributes are key-value pairs describing the data.

**property facets:** `Facets`

Facets are key-value pairs that were used to find this data.

**property ignore\_warnings:** `list[dict[str, Any]] | None`

Warnings to ignore when loading the data.

The `list` should contain `dict`'s with keyword arguments that will be passed to the `:func:`warnings.filterwarnings` function when calling the `to_iris` method.

**prepare()** → `None`

Prepare the data for access.

**Return type**

`None`

**to\_iris()** → `CubeList`

Load the data as Iris cubes.

**Returns**

The loaded data.

**Return type**

`iris.cube.CubeList`

```
esmvalcore.local.find_files(*, debug: bool = False, **facets: FacetValue) → list[LocalFile] |
tuple[list[LocalFile], list[Path]]
```

Find files on the local filesystem.

Deprecated since version 2.14.0: This function is deprecated and will be removed in version 2.16.0. Please use `esmvalcore.local.LocalDataSource.find_data()` instead.

The directories that are searched for files are defined in `esmvalcore.config.CFG` under the `'rootpath'` key using the directory structure defined under the `'drs'` key. If `esmvalcore.config.CFG['rootpath']` contains a key that matches the value of the project facet, those paths will be used. If there is no project specific key, the directories in `esmvalcore.config.CFG['rootpath']['default']` will be searched.

See *Input data* for extensive instructions on configuring ESMValCore so it can find files locally.

### Parameters

- **debug** (*bool*) – When debug is set to `True`, the function will return a tuple with the first element containing the files that were found and the second element containing the `glob.glob()` patterns that were used to search for files.
- **\*\*facets** (*FacetValue*) – Facets used to search for files. An `'*'` can be used to match any value. By default, only the latest version of a file will be returned. To select all versions use `version='*'`. It is also possible to specify multiple values for a facet, e.g. `exp=['historical', 'ssp585']` will match any file that belongs to either the historical or ssp585 experiment. The `timerange` facet can be specified in [ISO 8601 format](#).

### Return type

`list[LocalFile] | tuple[list[LocalFile], list[Path]]`

#### Note

A value of `timerange='*'` is supported, but combining a `'*'` with a time or period *as supported in the recipe* is currently not supported and will return all found files.

### Examples

Search for files containing surface air temperature from any CMIP6 model for the historical experiment:

```
>>> esmvalcore.local.find_files(
... project='CMIP6',
... activity='CMIP',
... mip='Amon',
... short_name='tas',
... exp='historical',
... dataset='*',
... ensemble='*',
... grid='*',
... institute='*',
...)
[LocalFile('/home/bandela/climate_data/CMIP6/CMIP/BCC/BCC-ESM1/historical/r1i1p1f1/
↪Amon/tas/gn/v20181214/tas_Amon_BCC-ESM1_historical_r1i1p1f1_gn_185001-201412.nc')]
```

### Returns

The files that were found.

### Return type

`list[LocalFile]`

### Parameters

- **debug** (*bool*)
- **facets** (*FacetValue*)

## 9.9 Preprocessor functions

All preprocessor functions are designed to preserve floating point data types. For example, input data of type `float32` will give output of `float32`. However, other data types may change, e.g., data of type `int` may give output of type `float64`.

```

esmvalcore.preprocessor.DEFAULT_ORDER: tuple[str, ...] = ('fix_file', 'load',
'fix_metadata', 'concatenate', 'cmor_check_metadata', 'clip_timerange', 'fix_data',
'cmor_check_data', 'add_supplementary_variables', 'derive', 'align_metadata',
'extract_time', 'extract_season', 'extract_month', 'resample_hours', 'resample_time',
'extract_levels', 'extract_surface_from_atm', 'weighting_landsea_fraction',
'mask_landsea', 'mask_glaciated', 'mask_landseaice', 'regrid',
'extract_coordinate_points', 'extract_point', 'extract_location', 'mask_multimodel',
'mask_fillvalues', 'mask_above_threshold', 'mask_below_threshold', 'mask_inside_range',
'mask_outside_range', 'clip', 'rolling_window_statistics', 'cumulative_sum',
'extract_region', 'extract_shape', 'extract_volume', 'extract_trajectory',
'extract_transect', 'detrend', 'extract_named_regions', 'axis_statistics',
'depth_integration', 'area_statistics', 'volume_statistics', 'local_solar_time',
'amplitude', 'zonal_statistics', 'meridional_statistics', 'accumulate_coordinate',
'hourly_statistics', 'daily_statistics', 'monthly_statistics', 'seasonal_statistics',
'annual_statistics', 'decadal_statistics', 'climate_statistics', 'anomalies',
'regrid_time', 'timeseries_filter', 'linear_trend', 'linear_trend_stderr',
'convert_units', 'histogram', 'ensemble_statistics', 'multi_model_statistics', 'bias',
'distance_metric', 'remove_supplementary_variables', 'save')

```

By default, preprocessor functions are applied in this order.

Preprocessor module.

### Functions:

|                                                           |                                                                    |
|-----------------------------------------------------------|--------------------------------------------------------------------|
| <i>accumulate_coordinate</i> (cube, coordinate)           | Weight data using the bounds from a given coordinate.              |
| <i>add_supplementary_variables</i> (cube, ...)            | Add ancillary variables and/or cell measures to cube (in-place).   |
| <i>align_metadata</i> (cube, target_project, ..., ...)    | Set cube metadata to entries from a specific target project.       |
| <i>amplitude</i> (cube, coords)                           | Calculate amplitude of cycles by aggregating over coordinates.     |
| <i>annual_statistics</i> (cube[, operator, ...])          | Compute annual statistics.                                         |
| <i>anomalies</i> (cube, period[, reference, ...])         | Compute anomalies using a mean with the specified granularity.     |
| <i>area_statistics</i> (cube, operator[, normalize])      | Apply a statistical operator in the horizontal plane.              |
| <i>axis_statistics</i> (cube, axis, operator[, ...])      | Perform statistics along a given axis.                             |
| <i>bias</i> (products[, reference, bias_type, ...])       | Calculate biases relative to a reference dataset.                  |
| <i>climate_statistics</i> (cube[, operator, period, ...]) | Compute climate statistics with the specified granularity.         |
| <i>clip</i> (cube[, minimum, maximum])                    | Clip values at a specified minimum and/or maximum value.           |
| <i>clip_timerange</i> (cube, timerange)                   | Extract time range with a resolution up to seconds.                |
| <i>cmor_check_data</i> (cube, cmor_table, mip, ...)       | Check if data conforms to variable's CMOR definition.              |
| <i>cmor_check_metadata</i> (cube, cmor_table, mip, ...)   | Check if metadata conforms to variable's CMOR definition.          |
| <i>concatenate</i> (cubes[, check_level])                 | Concatenate all cubes after fixing metadata.                       |
| <i>convert_units</i> (cube, units)                        | Convert the units of a cube to new ones (in-place).                |
| <i>cumulative_sum</i> (cube, coord[, weights, method])    | Calculate cumulative sum of the elements along a given coordinate. |
| <i>daily_statistics</i> (cube[, operator, ...])           | Compute daily statistics.                                          |
| <i>decadal_statistics</i> (cube[, operator, ...])         | Compute decadal statistics.                                        |
| <i>depth_integration</i> (cube)                           | Determine the total sum over the vertical component.               |
| <i>derive</i> (cubes, short_name, long_name, units)       | Derive variable.                                                   |
| <i>detrend</i> (cube[, dimension, method])                | Detrend data along a given dimension.                              |

continues on next page

Table 82 – continued from previous page

|                                                           |                                                                         |
|-----------------------------------------------------------|-------------------------------------------------------------------------|
| <i>distance_metric</i> (products, metric[, ...])          | Calculate distance metrics.                                             |
| <i>ensemble_statistics</i> (products, statistics, ...)    | Compute ensemble statistics.                                            |
| <i>extract_coordinate_points</i> (cube, definition, ...)  | Extract points from any coordinate with interpolation.                  |
| <i>extract_levels</i> (cube, levels, scheme[, ...])       | Perform vertical interpolation.                                         |
| <i>extract_location</i> (cube, location, scheme)          | Extract a point using a location name, with interpolation.              |
| <i>extract_month</i> (cube, month)                        | Slice cube to get only the data belonging to a specific month.          |
| <i>extract_named_regions</i> (cube, regions)              | Extract a specific named region.                                        |
| <i>extract_point</i> (cube, latitude, longitude, scheme)  | Extract a point, with interpolation.                                    |
| <i>extract_region</i> (cube, start_longitude, ...)        | Extract a region from a cube.                                           |
| <i>extract_season</i> (cube, season)                      | Slice cube to get only the data belonging to a specific season.         |
| <i>extract_shape</i> (cube, shapefile[, method, ...])     | Extract a region defined by a shapefile using masking.                  |
| <i>extract_surface_from_atm</i> (cube)                    | Extract surface from 3D atmospheric variable based on surface pressure. |
| <i>extract_time</i> (cube, start_year, start_month, ...)  | Extract a time range from a cube.                                       |
| <i>extract_trajectory</i> (cube, latitudes, longitudes)   | Extract data along a trajectory.                                        |
| <i>extract_transect</i> (cube[, latitude, longitude])     | Extract data along a line of constant latitude or longitude.            |
| <i>extract_volume</i> (cube, z_min, z_max[, ...])         | Subset a cube based on a range of values in the z-coordinate.           |
| <i>fix_data</i> (cube, short_name, project, dataset, mip) | Fix cube data if fixes are required.                                    |
| <i>fix_file</i> (file, short_name, project, dataset, ...) | Fix files before loading them into a <code>CubeList</code> .            |
| <i>fix_metadata</i> (cubes, short_name, project, ...)     | Fix cube metadata if fixes are required.                                |
| <i>histogram</i> (cube[, coords, bins, bin_range, ...])   | Calculate histogram.                                                    |
| <i>hourly_statistics</i> (cube, hours[, operator, ...])   | Compute hourly statistics.                                              |
| <i>linear_trend</i> (cube[, coordinate])                  | Calculate linear trend of data along a given coordinate.                |
| <i>linear_trend_stderr</i> (cube[, coordinate])           | Calculate standard error of linear trend along a given coordinate.      |
| <i>load</i> (file[, ignore_warnings, backend_kwargs])     | Load Iris cubes.                                                        |
| <i>local_solar_time</i> (cube)                            | Convert UTC time coordinate to local solar time (LST).                  |
| <i>mask_above_threshold</i> (cube, threshold)             | Mask above a specific threshold value.                                  |
| <i>mask_below_threshold</i> (cube, threshold)             | Mask below a specific threshold value.                                  |
| <i>mask_fillvalues</i> (products, threshold_fraction)     | Compute and apply a multi-dataset fillvalues mask.                      |
| <i>mask_glaciated</i> (cube[, mask_out])                  | Mask out glaciated areas.                                               |
| <i>mask_inside_range</i> (cube, minimum, maximum)         | Mask inside a specific threshold range.                                 |
| <i>mask_landsea</i> (cube, mask_out)                      | Mask out either land mass or sea (oceans, seas and lakes).              |
| <i>mask_landseaice</i> (cube, mask_out)                   | Mask out either landsea (combined) or ice.                              |
| <i>mask_multimodel</i> (products)                         | Apply common mask to all datasets (using logical OR).                   |
| <i>mask_outside_range</i> (cube, minimum, maximum)        | Mask outside a specific threshold range.                                |
| <i>meridional_statistics</i> (cube, operator[, ...])      | Compute meridional statistics.                                          |
| <i>monthly_statistics</i> (cube[, operator, ...])         | Compute monthly statistics.                                             |
| <i>multi_model_statistics</i> (products, span, ...)       | Compute multi-model statistics.                                         |
| <i>regrid</i> (cube, target_grid, scheme[, ...])          | Perform horizontal regridding.                                          |
| <i>regrid_time</i> (cube, frequency[, calendar, units])   | Align time coordinate for cubes.                                        |
| <i>remove_supplementary_variables</i> (cube)              | Remove supplementary variables from cube (in-place).                    |
| <i>resample_hours</i> (cube, interval[, offset, ...])     | Convert x-hourly data to y-hourly data.                                 |
| <i>resample_time</i> (cube[, month, day, hour])           | Change frequency of data by resampling it.                              |
| <i>rolling_window_statistics</i> (cube, coordinate, ...)  | Compute rolling-window statistics over a coordinate.                    |
| <i>save</i> (cubes, filename[, optimize_access, ...])     | Save iris cubes to file.                                                |
| <i>seasonal_statistics</i> (cube[, operator, ...])        | Compute seasonal statistics.                                            |
| <i>timeseries_filter</i> (cube, window, span[, ...])      | Apply a timeseries filter.                                              |

continues on next page

Table 82 – continued from previous page

|                                                             |                                              |
|-------------------------------------------------------------|----------------------------------------------|
| <code>volume_statistics(cube, operator[, normalize])</code> | Apply a statistical operation over a volume. |
| <code>weighting_landsea_fraction(cube, area_type)</code>    | Weight fields using land or sea fraction.    |
| <code>zonal_statistics(cube, operator[, normalize])</code>  | Compute zonal statistics.                    |

`esmvalcore.preprocessor.accumulate_coordinate(cube: Cube, coordinate: str | DimCoord | AuxCoord) → Cube`

Weight data using the bounds from a given coordinate.

The resulting cube will then have units given by `cube_units * coordinate_units`.

**Parameters**

- **cube** (*Cube*) – Data cube for the flux.
- **coordinate** (*str | DimCoord | AuxCoord*) – Name of the coordinate that will be used as weights.

**Returns**

Cube with the aggregated data.

**Return type**

`iris.cube.Cube`

**Raises**

- **ValueError** – If the coordinate is not found in the cube.
- **NotImplementedError** – If the coordinate is multidimensional.

`esmvalcore.preprocessor.add_supplementary_variables(cube: Cube, supplementary_cubes: Iterable[Cube]) → Cube`

Add ancillary variables and/or cell measures to cube (in-place).

**Parameters**

- **cube** (*Cube*) – Cube to add to.
- **supplementary\_cubes** (*Iterable[Cube]*) – Iterable of cubes containing the supplementary variables.

**Returns**

Cube with added ancillary variables and/or cell measures.

**Return type**

`iris.cube.Cube`

`esmvalcore.preprocessor.align_metadata(cube: Cube, target_project: str, target_mip: str, target_short_name: str, target_branding_suffix: str | None = None, strict: bool = True) → Cube`

Set cube metadata to entries from a specific target project.

This is useful to align variable metadata of different projects prior to performing multi-model operations (e.g., `multi_model_statistics()`). For example, standard names differ for some variables between CMIP5 and CMIP6 which would prevent the calculation of multi-model statistics between CMIP5 and CMIP6 data.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **target\_project** (*str*) – Project from which target metadata is read.
- **target\_mip** (*str*) – MIP table from which target metadata is read.

- **target\_short\_name** (*str*) – Variable short name from which target metadata is read.
- **target\_branding\_suffix** (*str* / *None*) – Branding suffix from which target metadata is read.
- **strict** (*bool*) – If True, raise an error if desired metadata cannot be read for variable `target_short_name` of MIP table `target_mip` and project `target_project`. If False, no error is raised.

**Returns**

Cube with updated metadata.

**Return type**

`iris.cube.Cube`

**Raises**

- **KeyError** – Invalid `target_project` given.
- **ValueError** – If `strict=True`: Variable `target_short_name` not available for MIP table `target_mip` of project `target_project`.

`esmvalcore.preprocessor.amplitude`(*cube, coords*)

Calculate amplitude of cycles by aggregating over coordinates.

**Note**

The amplitude is calculated as *peak-to-peak* amplitude (difference between maximum and minimum value of the signal). Other amplitude types are currently not supported.

**Parameters**

- **cube** (*iris.cube.Cube*) – Input data.
- **coords** (*str* or *list of str*) – Coordinates over which is aggregated. For example, use `'year'` to extract the annual cycle amplitude for each year in the data or `['day_of_year', 'year']` to extract the diurnal cycle amplitude for each individual day in the data. If the coordinates are not found in `cube`, try to add it via `iris.coord_categorisation` (at the moment, this only works for the temporal coordinates `day_of_month`, `day_of_year`, `hour`, `month`, `month_fullname`, `month_number`, `season`, `season_number`, `season_year`, `weekday`, `weekday_fullname`, `weekday_number` or `year`).

**Returns**

Amplitudes.

**Return type**

`iris.cube.Cube`

**Raises**

**`iris.exceptions.CoordinateNotFoundError`** – A coordinate is not found in `cube` and cannot be added via `iris.coord_categorisation`.

`esmvalcore.preprocessor.annual_statistics`(*cube: Cube, operator: str = 'mean', keep\_group\_coordinates: bool = False, \*\*operator\_kwargs: Any*) → *Cube*

Compute annual statistics.

Note that this function does not weight the annual mean if uneven time periods are present. Ie, all data inside the year are treated equally.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in [this table](#).
- **keep\_group\_coordinates** (*bool*) – If True, keep temporal group coordinates (i.e., year) in the result. If False, remove them.
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *operator*.

#### Returns

Annual statistics cube.

#### Return type

`iris.cube.Cube`

`esmvalcore.preprocessor.anomalies`(*cube: Cube, period: str, reference: dict | None = None, standardize: bool = False, relative: bool = False, seasons: Iterable[str] = ('DJF', 'MAM', 'JJA', 'SON')*) → `Cube`

Compute anomalies using a mean with the specified granularity.

Computes anomalies based on hourly, daily, monthly, seasonal or yearly means for the full available period.

#### Parameters

- **cube** (*Cube*) – Input cube.
- **period** (*str*) – Period to compute the statistic over. Available periods: *full, season, seasonal, monthly, month, mon, daily, day, hourly, hour, hr*.
- **reference** (*optional*) – Period of time to use a reference, as needed for the `extract_time()` preprocessor function. If `None`, all available data is used as a reference.
- **relative** (*optional*) – If True relative anomalies are calculated.
- **standardize** (*optional*) – If True standardized anomalies are calculated.
- **seasons** (*optional*) – Seasons to use if needed. Defaults to ('DJF', 'MAM', 'JJA', 'SON').

#### Returns

Anomalies cube.

#### Return type

`iris.cube.Cube`

`esmvalcore.preprocessor.area_statistics`(*cube: Cube, operator: str, normalize: Literal['subtract', 'divide'] | None = None, \*\*operator\_kwargs: Any*) → `Cube`

Apply a statistical operator in the horizontal plane.

We assume that the horizontal directions are ['longitude', 'latitude'].

[This table](#) shows a list of supported operators. All operators that support weights are by default weighted with the grid cell areas. Note that for area-weighted sums, the units of the resulting cube will be multiplied by  $m^2$ .

#### Parameters

- **cube** (*Cube*) – Input cube. The input cube should have a `iris.coords.CellMeasure` named 'cell\_area', unless it has regular 1D latitude and longitude coordinates so the cell areas can be computed using `iris.analysis.cartography.area_weights()`.
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in [this table](#).

- **normalize** (*Literal*['subtract', 'divide'] | *None*) – If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be *subtract* (statistics cube is subtracted from the input cube) or *divide* (input cube is divided by the statistics cube).
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *operator*.

**Returns**

Collapsed cube.

**Return type**

`iris.cube.Cube`

**Raises**

`iris.exceptions.CoordinateMultiDimError` – Cube has irregular or unstructured grid but supplementary variable *cell\_area* is not available.

`esmvalcore.preprocessor.axis_statistics`(*cube*: *Cube*, *axis*: *str*, *operator*: *str*, *normalize*: *Literal*['subtract', 'divide'] | *None* = *None*, **\*\*operator\_kwargs**: *Any*) → *Cube*

Perform statistics along a given axis.

Operates over an axis direction.

**Note**

The *mean*, *sum* and *rms* operations are weighted by the corresponding coordinate bounds by default. For *sum*, the units of the resulting cube will be multiplied by corresponding coordinate units.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **axis** (*str*) – Direction over where to apply the operator. Possible values are *x*, *y*, *z*, *t*.
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in *this table*.
- **normalize** (*Literal*['subtract', 'divide'] | *None*) – If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be *subtract* (statistics cube is subtracted from the input cube) or *divide* (input cube is divided by the statistics cube).
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *operator*.

**Returns**

Collapsed cube.

**Return type**

`iris.cube.Cube`

`esmvalcore.preprocessor.bias`(*products*: *set*[*PreprocessorFile*] | *Iterable*[*Cube*], *reference*: *Cube* | *None* = *None*, *bias\_type*: *BiasType* = 'absolute', *denominator\_mask\_threshold*: *float* = 0.001, *keep\_reference\_dataset*: *bool* = *False*) → *set*[*PreprocessorFile*] | *CubeList*

Calculate biases relative to a reference dataset.

The reference dataset needs to be broadcastable to all input *products*. This supports *iris*' rich broadcasting abilities. To ensure this, the preprocessors `esmvalcore.preprocessor.regrid()` and/or `esmvalcore.preprocessor.regrid_time()` might be helpful.

## Notes

The reference dataset can be specified with the *reference* argument. If *reference* is `None`, exactly one input dataset in the *products* set needs to have the facet `reference_for_bias: true` defined in the recipe. Please do **not** specify the option *reference* when using this preprocessor function in a recipe.

### Parameters

- **products** (`set[PreprocessorFile] | Iterable[Cube]`) – Input datasets/cubes for which the bias is calculated relative to a reference dataset/cube.
- **reference** (`Cube | None`) – Cube which is used as reference for the bias calculation. If `None`, *products* needs to be a `set` of `~esmvalcore.preprocessor.PreprocessorFile` objects and exactly one dataset in *products* needs the facet `reference_for_bias: true`. Do not specify this argument in a recipe.
- **bias\_type** (`BiasType`) – Bias type that is calculated. Must be one of `'absolute'` (dataset - ref) or `'relative'` ((dataset - ref) / ref).
- **denominator\_mask\_threshold** (`float`) – Threshold to mask values close to zero in the denominator (i.e., the reference dataset) during the calculation of relative biases. All values in the reference dataset with absolute value less than the given threshold are masked out. This setting is ignored when `bias_type` is set to `'absolute'`. Please note that for some variables with very small absolute values (e.g., carbon cycle fluxes, which are usually  $< 10^{-6} \text{ kg m}^{-2} \text{ s}^{-1}$ ) it is absolutely essential to change the default value in order to get reasonable results.
- **keep\_reference\_dataset** (`bool`) – If `True`, keep the reference dataset in the output. If `False`, drop the reference dataset. Ignored if *reference* is given.

### Returns

Output datasets/cubes. Will be a `set` of `PreprocessorFile` objects if *products* is also one, a `CubeList` otherwise.

### Return type

`set[PreprocessorFile] | CubeList`

### Raises

**ValueError** – Not exactly one input datasets contains the facet `reference_for_bias: true` if `reference=None`; `reference=None` and the input products are given as iterable of `Cube` objects; `bias_type` is not one of `'absolute'` or `'relative'`.

```
esmvalcore.preprocessor.climate_statistics(cube: Cube, operator: str = 'mean', period: str = 'full',
 seasons: Iterable[str] = ('DJF', 'MAM', 'JJA', 'SON'),
 **operator_kwargs: Any) → Cube
```

Compute climate statistics with the specified granularity.

Computes statistics for the whole dataset. It is possible to get them for the full period or with the data grouped by hour, day, month or season.

#### Note

The *mean*, *sum* and *rms* operations over the *full* period are weighted by the time coordinate, i.e., the length of the time intervals. For *sum*, the units of the resulting cube will be multiplied by corresponding time units (e.g., days).

If a period other than *full* is used, time points will be put into bins, which may shift existing time points. For example, for *period=hourly*, a time point at 01:30h will be moved to the corresponding full hour (01:00h).

### Parameters

- **cube** (*Cube*) – Input cube.
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in [this table](#).
- **period** (*str*) – Period to compute the statistic over. Available periods: *full*, *season*, *seasonal*, *monthly*, *month*, *mon*, *daily*, *day*, *hourly*, *hour*, *hr*.
- **seasons** (*Iterable[str]*) – Seasons to use if needed. Defaults to ('DJF', 'MAM', 'JJA', 'SON').
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *operator*.

### Returns

Climate statistics cube.

### Return type

`iris.cube.Cube`

`esmvalcore.preprocessor.clip(cube: Cube, minimum: float | None = None, maximum: float | None = None) → Cube`

Clip values at a specified minimum and/or maximum value.

Values lower than minimum are set to minimum and values higher than maximum are set to maximum.

### Parameters

- **cube** (*Cube*) – Input cube.
- **minimum** (*float | None*) – Lower threshold to be applied on input cube data.
- **maximum** (*float | None*) – Upper threshold to be applied on input cube data.

### Returns

Clipped cube.

### Return type

`iris.cube.Cube`

### Raises

**ValueError** – minimum and maximum are set to None.

`esmvalcore.preprocessor.clip_timerange(cube: Cube, timerange: str) → Cube`

Extract time range with a resolution up to seconds.

### Parameters

- **cube** (*Cube*) – Input cube.
- **timerange** (*str*) – Time range in ISO 8601 format.

### Returns

Sliced cube.

### Return type

`iris.cube.Cube`

### Raises

- **ValueError** – Time ranges are outside the cube's time limits.
- **isodate.isoerror.ISO8601Error** – Start/end times can not be parsed by isodate.

```
esmvalcore.preprocessor.cmor_check_data(cube: Cube, cmor_table: str, mip: str, short_name: str, *,
 branding_suffix: str | None = None, frequency: str | None =
 None, check_level: CheckLevels = CheckLevels.DEFAULT) →
 Cube
```

Check if data conforms to variable's CMOR definition.

### Parameters

- **cube** (*Cube*) – Data cube to check.
- **cmor\_table** (*str*) – CMOR definitions to use (i.e., the variable's project).
- **mip** (*str*) – Variable's MIP.
- **short\_name** (*str*) – Variable's short name
- **branding\_suffix** (*str* | *None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table. Used by the CMIP7 project.
- **frequency** (*str* | *None*) – Data frequency. If not given, use the one from the CMOR table of the variable.
- **check\_level** (*CheckLevels*) – Level of strictness of the checks.

### Returns

Checked cube.

### Return type

`iris.cube.Cube`

```
esmvalcore.preprocessor.cmor_check_metadata(cube: Cube, cmor_table: str, mip: str, short_name: str, *,
 branding_suffix: str | None = None, frequency: str | None =
 None, check_level: CheckLevels =
 CheckLevels.DEFAULT) → Cube
```

Check if metadata conforms to variable's CMOR definition.

None of the checks at this step will force the cube to load the data.

### Parameters

- **cube** (*Cube*) – Data cube to check.
- **cmor\_table** (*str*) – CMOR definitions to use (i.e., the variable's project).
- **mip** (*str*) – Variable's MIP.
- **short\_name** (*str*) – Variable's short name.
- **branding\_suffix** (*str* | *None*) – A suffix that will be appended to `short_name` when looking up the variable in the CMOR table. Used by the CMIP7 project.
- **frequency** (*str* | *None*) – Data frequency. If not given, use the one from the CMOR table of the variable.
- **check\_level** (*CheckLevels*) – Level of strictness of the checks.

### Returns

Checked cube.

**Return type**`iris.cube.Cube`

`esmvalcore.preprocessor.concatenate(cubes: Sequence[Cube], check_level: CheckLevels = CheckLevels.DEFAULT) → Cube`

Concatenate all cubes after fixing metadata.

**Parameters**

- **cubes** (*iterable of `iris.cube.Cube`*) – Data cubes to be concatenated
- **check\_level** (`CheckLevels`) – Level of strictness of the checks in the concatenation.

**Returns**

**cube** – Resulting concatenated cube.

**Return type**`iris.cube.Cube`**Raises**

**ValueError** – Concatenation was not possible.

`esmvalcore.preprocessor.convert_units(cube: Cube, units: str | Unit) → Cube`

Convert the units of a cube to new ones (in-place).

**Note**

Allows special unit conversions which transforms one quantity to another (physically related) quantity, which may also change the input cube's `standard_name`. These quantities are identified via their `standard_name` and their `units` (units convertible to the ones defined are also supported). For example, this enables conversions between precipitation fluxes measured in  $\text{kg m}^{-2} \text{s}^{-1}$  and precipitation rates measured in  $\text{mm day}^{-1}$  (and vice versa).

Currently, the following special conversions are supported:

- `precipitation_flux` ( $\text{kg m}^{-2} \text{s}^{-1}$ ) – `lwe_precipitation_rate` ( $\text{mm day}^{-1}$ )
- `water_evaporation_flux` ( $\text{kg m}^{-2} \text{s}^{-1}$ ) – `lwe_water_evaporation_rate` ( $\text{mm day}^{-1}$ )
- `water_potential_evaporation_flux` ( $\text{kg m}^{-2} \text{s}^{-1}$ ) – `None` ( $\text{mm day}^{-1}$ )
- `equivalent_thickness_at_stp_of_atmosphere_ozone_content` (m) – `equivalent_thickness_at_stp_of_atmosphere_ozone_content` (DU)
- `surface_air_pressure` (Pa) – `atmosphere_mass_of_air_per_unit_area` ( $\text{kg m}^{-2}$ )

Names in the list correspond to `standard_names` of the input data. Conversions are allowed from each quantity to any other quantity given in a bullet point. The corresponding target quantity is inferred from the desired target units. In addition, any other units convertible to the ones given are also supported (e.g., instead of  $\text{mm day}^{-1}$ ,  $\text{m s}^{-1}$  is also supported).

Note that for precipitation and evaporation variables, a water density of  $1000 \text{ kg m}^{-3}$  is assumed.

**Parameters**

- **cube** (`Cube`) – Input cube (modified in place).
- **units** (`str | Unit`) – New units.

**Returns**

Converted cube. Just returned for convenience; input cube is modified in place.

**Return type**

`iris.cube.Cube`

**Raises**

- `iris.exceptions.UnitConversionError` – Old units are unknown.
- `ValueError` – Old units are not convertible to new units.

`esmvalcore.preprocessor.cumulative_sum(cube: Cube, coord: Coord | str, weights: np.ndarray | da.Array | bool | None = None, method: Literal['sequential', 'blelloch'] = 'sequential') → Cube`

Calculate cumulative sum of the elements along a given coordinate.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **coord** (*Coord | str*) – Coordinate over which the cumulative sum is calculated. Must be 0D or 1D.
- **weights** (*np.ndarray | da.Array | bool | None*) – Weights for the calculation of the cumulative sum. Each element in the data is multiplied by the corresponding weight before summing. Can be an array of the same shape as the input data, `False` or `None` (no weighting), or `True` (calculate the weights from the coordinate bounds; only works if each coordinate point has exactly 2 bounds).
- **method** (*Literal['sequential', 'blelloch']*) – Method used to perform the cumulative sum. Only relevant if the cube has *lazy data*. See `dask.array.cumsum()` for details.

**Returns**

Cube of cumulative sum. Has same dimensions and coordinates of the input cube.

**Return type**

`Cube`

**Raises**

- `iris.exceptions.CoordinateMultiDimError` – coord is not 0D or 1D.
- `iris.exceptions.CoordinateNotFoundError` – coord is not found in cube.

`esmvalcore.preprocessor.daily_statistics(cube: Cube, operator: str = 'mean', keep_group_coordinates: bool = False, **operator_kwargs: Any) → Cube`

Compute daily statistics.

Chunks time in daily periods and computes statistics over them;

**Parameters**

- **cube** (*Cube*) – Input cube.
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in *this table*.
- **keep\_group\_coordinates** (*bool*) – If `True`, keep temporal group coordinates (i.e., `day_of_year`, `year`) in the result. If `False`, remove them.
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *operator*.

**Returns**

Daily statistics cube.

**Return type**`iris.cube.Cube`

`esmvalcore.preprocessor.decadal_statistics(cube: Cube, operator: str = 'mean', keep_group_coordinates: bool = False, **operator_kwargs: Any) → Cube`

Compute decadal statistics.

Note that this function does not weight the decadal mean if uneven time periods are present. Ie, all data inside the decade are treated equally.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in *this table*.
- **keep\_group\_coordinates** (*bool*) – If True, keep temporal group coordinates (i.e., decade) in the result. If False, remove them.
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *operator*.

**Returns**

Decadal statistics cube.

**Return type**`iris.cube.Cube`

`esmvalcore.preprocessor.depth_integration(cube: Cube) → Cube`

Determine the total sum over the vertical component.

Requires a 3D cube. The z-coordinate integration is calculated by taking the sum in the z direction of the cell contents multiplied by the cell thickness. The units of the resulting cube are multiplied by the z-coordinate units.

**Parameters**

**cube** (*Cube*) – Input cube.

**Returns**

Collapsed cube.

**Return type**`iris.cube.Cube`

`esmvalcore.preprocessor.derive(cubes: CubeList, short_name: str, long_name: str, units: str | Unit, standard_name: str | None = None) → Cube`

Derive variable.

**Parameters**

- **cubes** (*CubeList*) – Includes all the needed variables for derivation defined in `get_required()`.
- **short\_name** (*str*) – short\_name
- **long\_name** (*str*) – long\_name
- **units** (*str* | *Unit*) – units
- **standard\_name** (*str* | *None*) – standard\_name

**Returns**

The new derived variable.

**Return type**`iris.cube.Cube``esmvalcore.preprocessor.detrend(cube, dimension='time', method='linear')`

Detrend data along a given dimension.

**Parameters**

- **cube** (`iris.cube.Cube`) – input cube.
- **dimension** (`str`) – Dimension to detrend
- **method** (`str`) – Method to detrend. Available: linear, constant. See documentation of 'scipy.signal.detrend' for details

**Returns**

Detrended cube

**Return type**`iris.cube.Cube`

`esmvalcore.preprocessor.distance_metric(products: set[PreprocessorFile] | Iterable[Cube], metric: MetricType, reference: Cube | None = None, coords: Iterable[Coord] | Iterable[str] | None = None, keep_reference_dataset: bool = True, **kwargs: Any) → set[PreprocessorFile] | CubeList`

Calculate distance metrics.

All input datasets need to have identical dimensional coordinates. This can for example be ensured with the preprocessors `esmvalcore.preprocessor.regrid()` and/or `esmvalcore.preprocessor.regrid_time()`.

**Notes**

This preprocessor requires a reference dataset, which can be specified with the *reference* argument. If *reference* is `None`, exactly one input dataset in the *products* set needs to have the facet `reference_for_metric: true` defined in the recipe. Please do **not** specify the option *reference* when using this preprocessor function in a recipe.

**Parameters**

- **products** (`set[PreprocessorFile] | Iterable[Cube]`) – Input datasets/cubes for which the distance metric is calculated relative to a reference dataset/cube.
- **metric** (`MetricType`) – Distance metric that is calculated. Must be one of
  - 'rmse': Unweighted root mean square error
  - 'weighted\_rmse': Weighted root mean square error
  - 'pearsonr': Unweighted Pearson correlation coefficient
  - 'weighted\_pearsonr': Weighted Pearson correlation coefficient
  - 'emd': Unweighted Earth mover's distance
  - 'weighted\_emd': Weighted Earth mover's distance

The Earth mover's distance is also known as first Wasserstein metric  $W_1$ .

A detailed description of these metrics can be found [here](#).

**Note**

Metrics starting with *weighted\_* will calculate weighted distance metrics if possible. Currently, the following *coords* (or any combinations that include them) will trigger weighting: *time* (will use lengths of time intervals as weights) and *latitude* (will use cell area weights). Time weights are always calculated from the input data. Area weights can be given as supplementary variables to the recipe (*areacella* or *areacello*, see [Defining supplementary variables \(ancillary variables and cell measures\)](#)) or calculated from the input data (this only works for regular grids). By default, **NO** supplementary variables will be used; they need to be explicitly requested in the recipe.

- **reference** (*Cube* | *None*) – Cube which is used as reference for the distance metric calculation. If *None*, *products* needs to be a [set](#) of `PreprocessorFile` objects and exactly one dataset in *products* needs the facet `reference_for_metric: true`. Do not specify this argument in a recipe.
- **coords** (*Iterable[Coord]* | *Iterable[str]* | *None*) – Coordinates over which the distance metric is calculated. If *None*, calculate the metric over all coordinates, which results in a scalar cube.
- **keep\_reference\_dataset** (*bool*) – If `True`, also calculate the distance of the reference dataset with itself. If `False`, drop the reference dataset.
- **\*\*kwargs** (*Any*) – Additional options for the metric calculation. The following keyword arguments are supported:
  - *rmse* and *weighted\_rmse*: none.
  - *pearsonr* and *weighted\_pearsonr*: `mdtol`, `common_mask` (all keyword arguments are passed to `iris.analysis.stats.pearsonr()`, see that link for more details on these arguments). Note: in contrast to `pearsonr()`, `common_mask=True` by default.
  - *emd* and *weighted\_emd*: `n_bins` = number of bins used to create discrete probability distribution of data before calculating the EMD (`int`, default: 100).

**Returns**

Output datasets/cubes. Will be a [set](#) of `PreprocessorFile` objects if *products* is also one, a `CubeList` otherwise.

**Return type**

`set` of `esmvalcore.preprocessor.PreprocessorFile` or `iris.cube.CubeList`

**Raises**

- **ValueError** – Shape and coordinates of *products* and reference data does not match; not exactly one input datasets contains the facet `reference_for_metric: true` if `reference=None`; `reference=None` and the input *products* are given as iterable of `Cube` objects; an invalid `metric` has been given.
- **iris.exceptions.CoordinateNotFoundError** – *longitude* is not found in cube if a weighted metric shall be calculated, *latitude* is in *coords*, and no *cell\_area* is given as [Defining supplementary variables \(ancillary variables and cell measures\)](#).

```
esmvalcore.preprocessor.ensemble_statistics(products: set[PreprocessorFile] | Iterable[Cube],
 statistics: list[str | dict], output_products: dict[str,
 PreprocessorFile] | None, span: str = 'overlap',
 ignore_scalar_coords: bool = False) → dict | set
```

Compute ensemble statistics.

An ensemble grouping is performed on the input products (using the *ensemble* facet of input datasets). The statistics are then computed calling `esmvalcore.preprocessor.multi_model_statistics()` with appropriate groups.

### Parameters

- **products** (`set[PreprocessorFile] | Iterable[Cube]`) – Cubes (or products) over which the statistics will be computed.
- **statistics** (`list[str | dict]`) – Statistical operations to be computed, e.g., ['mean', 'median']. For some statistics like percentiles, it is also possible to pass additional keyword arguments, e.g., [{'operator': 'percentile', 'percent': 20}]. All supported options are given in [this table](#).
- **output\_products** (`dict[str, PreprocessorFile] | None`) – For internal use only. A dict with statistics names as keys and preprocessorfiles as values. If products are passed as input, the statistics cubes will be assigned to these output products.
- **span** (`str`) – Overlap or full; if overlap, statistics are computed on common time- span; if full, statistics are computed on full time spans, ignoring missing data.
- **ignore\_scalar\_coords** (`bool`) – If True, remove any scalar coordinate in the input datasets before merging the input cubes into the multi-dataset cube. The resulting multi-dataset cube will have no scalar coordinates (the actual input datasets will remain unchanged). If False, scalar coordinates will remain in the input datasets, which might lead to merge conflicts in case the input datasets have different scalar coordinates.

### Returns

A `dict` of cubes or `set` of `output_products` depending on the type of `products`.

### Return type

`dict | set`

### ➔ See also

`esmvalcore.preprocessor.multi_model_statistics()`, the

`esmvalcore.preprocessor.extract_coordinate_points(cube: Cube, definition: dict[str, ArrayLike], scheme: NamedPointInterpolationScheme) → Cube`

Extract points from any coordinate with interpolation.

Multiple points can also be extracted, by supplying an array of coordinates. The resulting point cube will match the respective coordinates to those of the input coordinates. If the input coordinate is a scalar, the dimension will be a scalar in the output cube.

### Parameters

- **cube** (`Cube`) – The source cube to extract a point from.
- **definition** (`dict[str, ArrayLike]`) – The coordinate - values pairs to extract
- **scheme** (`NamedPointInterpolationScheme`) – The interpolation scheme. ‘linear’ or ‘nearest’. No default.

### Returns

Returns a cube with the extracted point(s), and with adjusted latitude and longitude coordinates (see above). If desired point outside values for at least one coordinate, this cube will have fully masked data.

**Return type**`iris.cube.Cube`**Raises****ValueError**: – If the interpolation scheme is not provided or is not recognised.

`esmvalcore.preprocessor.extract_levels`(*cube*: *Cube*, *levels*: *ArrayLike* | *Array*, *scheme*: *Literal*['linear', 'nearest', 'linear\_extrapolate', 'nearest\_extrapolate'], *coordinate*: *str* | *None* = *None*, *rtol*: *float* = *1e-07*, *atol*: *float* | *None* = *None*)  
→ *Cube*

Perform vertical interpolation.

**Parameters**

- **cube** (*Cube*) – The source cube to be vertically interpolated.
- **levels** (*ArrayLike* | *Array*) – One or more target levels for the vertical interpolation. Assumed to be in the same S.I. units of the source cube vertical dimension coordinate. If the requested levels are sufficiently close to the levels of the cube, cube slicing will take place instead of interpolation.
- **scheme** (*Literal*['linear', 'nearest', 'linear\_extrapolate', 'nearest\_extrapolate']) – The vertical interpolation scheme to use. Choose from 'linear', 'nearest', 'linear\_extrapolate', 'nearest\_extrapolate'.
- **coordinate** (*str* | *None*) – The coordinate to interpolate. If specified, pressure levels (if present) can be converted to height levels and vice versa using the US standard atmosphere. E.g. 'coordinate = altitude' will convert existing pressure levels (air\_pressure) to height levels (altitude); 'coordinate = air\_pressure' will convert existing height levels (altitude) to pressure levels (air\_pressure).
- **rtol** (*float*) – Relative tolerance for comparing the levels in *cube* to the requested levels. If the levels are sufficiently close, the requested levels will be assigned to the cube and no interpolation will take place.
- **atol** (*float* | *None*) – Absolute tolerance for comparing the levels in *cube* to the requested levels. If the levels are sufficiently close, the requested levels will be assigned to the cube and no interpolation will take place. By default, *atol* will be set to  $10^{-7}$  times the mean value of the levels on the cube.

**Returns**

A cube with the requested vertical levels.

**Return type**`iris.cube.Cube`**➔ See also*****regrid***

Perform horizontal regridding.

`esmvalcore.preprocessor.extract_location`(*cube*: *Cube*, *location*: *str*, *scheme*: *Literal*['linear', 'nearest'])  
→ *Cube*

Extract a point using a location name, with interpolation.

Extracts a single location point from a cube, according to the interpolation scheme *scheme*.

The function just retrieves the coordinates of the location and then calls the `extract_point` preprocessor.

It can be used to locate cities and villages, but also mountains or other geographical locations.

**Note**

The geolocator needs a working internet connection.

**Parameters**

- **cube** (*Cube*) – The source cube to extract a point from.
- **location** (*str*) – The reference location. Examples: 'mount everest', 'romania', 'new york, usa'
- **scheme** (*Literal*['linear', 'nearest']) – The interpolation scheme. 'linear' or 'nearest'. No default.

**Returns**

Returns a cube with the extracted point, and with adjusted latitude and longitude coordinates.

**Return type**

`iris.cube.Cube`

**Raises**

- **ValueError**: – If location is not supplied as a preprocessor parameter.
- **ValueError**: – If scheme is not supplied as a preprocessor parameter.
- **ValueError**: – If given location cannot be found by the geolocator.

`esmvalcore.preprocessor.extract_month(cube: Cube, month: int) → Cube`

Slice cube to get only the data belonging to a specific month.

**Parameters**

- **cube** (*Cube*) – Original data
- **month** (*int*) – Month to extract as a number from 1 to 12.

**Returns**

Cube for specified month.

**Return type**

`iris.cube.Cube`

**Raises**

**ValueError** – Requested month is not present in the cube.

`esmvalcore.preprocessor.extract_named_regions(cube: Cube, regions: str | Iterable[str]) → Cube`

Extract a specific named region.

The region coordinate exist in certain CMIP datasets. This preprocessor allows a specific named regions to be extracted.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **regions** (*str | Iterable[str]*) – A region or list of regions to extract.

**Returns**

Smaller cube.

**Return type**`iris.cube.Cube`**Raises**

- **ValueError** – regions is not list or tuple or set.
- **ValueError** – region not included in cube.

`esmvalcore.preprocessor.extract_point`(*cube: Cube, latitude: ArrayLike, longitude: ArrayLike, scheme: NamedPointInterpolationScheme*) → *Cube*

Extract a point, with interpolation.

Extracts a single latitude/longitude point from a cube, according to the interpolation scheme *scheme*.

Multiple points can also be extracted, by supplying an array of latitude and/or longitude coordinates. The resulting point cube will match the respective latitude and longitude coordinate to those of the input coordinates. If the input coordinate is a scalar, the dimension will be missing in the output cube (that is, it will be a scalar).

If the point to be extracted has at least one of the coordinate point values outside the interval of the cube's same coordinate values, then no extrapolation will be performed, and the resulting extracted cube will have fully masked data.

**Parameters**

- **cube** (*Cube*) – The source cube to extract a point from.
- **latitude** (*ArrayLike*) – The latitude of the point.
- **longitude** (*ArrayLike*) – The longitude of the point.
- **scheme** (*NamedPointInterpolationScheme*) – The interpolation scheme. 'linear' or 'nearest'. No default.

**Returns**

Returns a cube with the extracted point(s), and with adjusted latitude and longitude coordinates (see above). If desired point outside values for at least one coordinate, this cube will have fully masked data.

**Return type**`iris.cube.Cube`**Raises**

**ValueError:** – If the interpolation scheme is None or unrecognized.

**Examples**

With a cube that has the coordinates

- latitude: [1, 2, 3, 4]
- longitude: [1, 2, 3, 4]
- data values: [[[1, 2, 3, 4], [5, 6, ...], [...], [...], ... ]]

```
>>> point = extract_point(cube, 2.5, 2.5, 'linear')
>>> point.data
array([8.5, 24.5, 40.5, 56.5])
```

Extraction of multiple points at once, with a nearest matching scheme. The values for 0.1 will result in masked values, since this lies outside the cube grid.

```

>>> point = extract_point(cube, [1.4, 2.1], [0.1, 1.1],
... 'nearest')
>>> point.data.shape
(4, 2, 2)
>>> # x, y, z indices of masked values
>>> np.where(~point.data.mask)
(array([0, 0, 1, 1, 2, 2, 3, 3]), array([0, 1, 0, 1, 0, 1, 0, 1]),
array([1, 1, 1, 1, 1, 1, 1, 1]))
>>> point.data[~point.data.mask].data
array([1, 5, 17, 21, 33, 37, 49, 53])

```

`esmvalcore.preprocessor.extract_region`(*cube*: *Cube*, *start\_longitude*: *float*, *end\_longitude*: *float*, *start\_latitude*: *float*, *end\_latitude*: *float*) → *Cube*

Extract a region from a cube.

Function that subsets a cube on a box (*start\_longitude*, *end\_longitude*, *start\_latitude*, *end\_latitude*).

#### Parameters

- **cube** (*Cube*) – Input data cube.
- **start\_longitude** (*float*) – Western boundary longitude.
- **end\_longitude** (*float*) – Eastern boundary longitude.
- **start\_latitude** (*float*) – Southern Boundary latitude.
- **end\_latitude** (*float*) – Northern Boundary Latitude.

#### Returns

Smaller cube.

#### Return type

`iris.cube.Cube`

`esmvalcore.preprocessor.extract_season`(*cube*: *Cube*, *season*: *str*) → *Cube*

Slice cube to get only the data belonging to a specific season.

#### Parameters

- **cube** (*Cube*) – Original data
- **season** (*str*) – Season to extract. Available: DJF, MAM, JJA, SON and all sequentially correct combinations: e.g. JJAS

#### Returns

data cube for specified season.

#### Return type

`iris.cube.Cube`

#### Raises

**ValueError** – Requested season is not present in the cube.

`esmvalcore.preprocessor.extract_shape`(*cube*: *Cube*, *shapefile*: *str* | *Path*, *method*: *str* = 'contains', *crop*: *bool* = *True*, *decomposed*: *bool* = *False*, *ids*: *list* | *dict* | *None* = *None*) → *Cube*

Extract a region defined by a shapefile using masking.

Note that this function does not work for shapes crossing the prime meridian or poles.

#### Parameters

- **cube** (*Cube*) – Input cube.
- **shapefile** (*str* | *Path*) – A shapefile defining the region(s) to extract. Also accepts the following strings to load special shapefiles:
  - 'ar6': IPCC WG1 reference regions (v4) used in Assessment Report 6 (<https://doi.org/10.5281/zenodo.5176260>). Should be used in combination with a `dict` for the argument *ids*, e.g., `ids={'Acronym': ['GIC', 'WNA']}`.
- **method** (*str*) – Select all points contained by the shape or select a single representative point. Choose either 'contains' or 'representative'. If 'contains' is used, but not a single grid point is contained by the shape, a representative point will be selected.
- **crop** (*bool*) – In addition to masking, crop the resulting cube using `extract_region()`. Data on irregular grids will not be cropped.
- **decomposed** (*bool*) – If set to `True`, the output cube will have an additional dimension *shape\_id* describing the requested regions.
- **ids** (*list* | *dict* | *None*) – Shapes to be read from the shapefile. Can be given as:
  - **list**: IDs are assigned from the attributes *name*, *NAME*, *Name*, *id*, or *ID* (in that priority order; the first one available is used). If none of these attributes are available in the shapefile, assume that the given *ids* correspond to the reading order of the individual shapes, e.g., `ids=[0, 2]` corresponds to the first and third shape read from the shapefile. Note: An empty list is interpreted as `ids=None`.
  - **dict**: IDs (dictionary value; **list** of *str*) are assigned from attribute given as dictionary key (*str*). Only dictionaries with length 1 are supported. Example: `ids={'Acronym': ['GIC', 'WNA']}` for `shapefile='ar6'`.
  - *None*: select all available shapes from the shapefile.

**Returns**

Cube containing the extracted region.

**Return type**

`iris.cube.Cube`

 **See also**
**`extract_region`**

Extract a region from a cube.

`esmvalcore.preprocessor.extract_surface_from_atm(cube: Cube) → Cube`

Extract surface from 3D atmospheric variable based on surface pressure.

**Parameters**

**cube** (*Cube*) – Input cube. Needs `AncillaryVariable` `surface_air_pressure`.

**Returns**

Collapsed cube.

**Return type**

`iris.cube.Cube`

`esmvalcore.preprocessor.extract_time(cube: Cube, start_year: int | None, start_month: int, start_day: int, end_year: int | None, end_month: int, end_day: int) → Cube`

Extract a time range from a cube.

Given a time range passed in as a series of years, months and days, it returns a time-extracted cube with data only within the specified time range.

#### Parameters

- **cube** (*Cube*) – Input cube.
- **start\_year** (*int* | *None*) – Start year. If *None*, the date ranges (*start\_month-start\_day* to *end\_month-end\_day*) are selected in each year. For example, ranges Feb 3 - Apr 6 in each year are selected if *start\_year=None*, *start\_month=2*, *start\_day=3*, *end\_year=None*, *end\_month=4*, *end\_day=6*. If *start\_year* is *None*, *end\_year* has to be *None* too.
- **start\_month** (*int*) – Start month.
- **start\_day** (*int*) – Start day.
- **end\_year** (*int* | *None*) – End year. If *None*, the date ranges (*start\_month-start\_day* to *end\_month-end\_day*) are selected in each year. For example, ranges Feb 3 - Apr 6 in each year are selected if *start\_year=None*, *start\_month=2*, *start\_day=3*, *end\_year=None*, *end\_month=4*, *end\_day=6*. If *end\_year* is *None*, *start\_year* has to be *None* too.
- **end\_month** (*int*) – End month.
- **end\_day** (*int*) – End day.

#### Returns

Sliced cube.

#### Return type

`iris.cube.Cube`

#### Raises

**ValueError** – Time ranges are outside the cube time limits.

`esmvalcore.preprocessor.extract_trajectory(cube: Cube, latitudes: Sequence[float], longitudes: Sequence[float], number_points: int = 2) → Cube`

Extract data along a trajectory.

latitudes and longitudes are the pairs of coordinates for two points. `number_points` is the number of points between the two points.

This version uses the expensive interpolate method, but it may be necessary for irregular grids.

If only two latitude and longitude coordinates are given, `extract_trajectory` will produce a cube will extrapolate along a line between those two points, and will add `number_points` points between the two corners.

If more than two points are provided, then `extract_trajectory` will produce a cube which has extrapolated the data of the cube to those points, and `number_points` is not needed.

#### Parameters

- **cube** (*Cube*) – Input cube.
- **latitudes** (*Sequence[float]*) – Latitude coordinates.
- **longitudes** (*Sequence[float]*) – Longitude coordinates.
- **number\_points** (*optional*) – Number of points to extrapolate.

#### Returns

Collapsed cube.

#### Return type

`iris.cube.Cube`

**Raises**

**ValueError** – Latitude and longitude have different dimensions.

```
esmvalcore.preprocessor.extract_transect(cube: Cube, latitude: None | float | Iterable[float] = None,
 longitude: None | float | Iterable[float] = None) → Cube
```

Extract data along a line of constant latitude or longitude.

Both arguments, latitude and longitude, are treated identically. Either argument can be a single float, or a pair of floats, or can be left empty. The single float indicates the latitude or longitude along which the transect should be extracted. A pair of floats indicate the range that the transect should be extracted along the secondary axis.

For instance `'extract_transect(cube, longitude=-28)'` will produce a transect along 28 West.

Also, `'extract_transect(cube, longitude=-28, latitude=[-50, 50])'` will produce a transect along 28 West between 50 south and 50 North.

This function is not yet implemented for irregular arrays - instead try the `extract_trajectory` function, but note that it is currently very slow. Alternatively, use the `regrid` preprocessor to regrid along a regular grid and then extract the transect.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **latitude** (*optional*) – Transect latitude or range.
- **longitude** (*optional*) – Transect longitude or range.

**Returns**

Collapsed cube.

**Return type**

`iris.cube.Cube`

**Raises**

- **ValueError** – Slice extraction not implemented for irregular grids.
- **ValueError** – Latitude and longitude are both floats or lists; not allowed to slice on both axes at the same time.

```
esmvalcore.preprocessor.extract_volume(cube: Cube, z_min: float, z_max: float, interval_bounds: str =
 'open', nearest_value: bool = False) → Cube
```

Subset a cube based on a range of values in the z-coordinate.

Function that subsets a cube on a box of  $(z\_min, z\_max)$ ,  $(z\_min, z\_max]$ ,  $[z\_min, z\_max)$  or  $[z\_min, z\_max]$ . Note that this requires the requested z-coordinate range to be the same sign as the iris cube. ie, if the cube has z-coordinate as negative, then `z_min` and `z_max` need to be negative numbers. If `nearest_value` is set to `False`, the extraction will be performed with the given `z_min` and `z_max` values. If `nearest_value` is set to `True`, the cube extraction will be performed taking into account the `z_coord` values that are closest to the `z_min` and `z_max` values.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **z\_min** (*float*) – Minimum depth to extract.
- **z\_max** (*float*) – Maximum depth to extract.
- **interval\_bounds** (*str*) – Sets left bound of the interval to either 'open', 'closed', 'left\_closed' or 'right\_closed'.

- **nearest\_value** (*bool*) – Extracts considering the nearest value of z-coord to z\_min and z\_max.

**Returns**

z-coord extracted cube.

**Return type**

`iris.cube.Cube`

`esmvalcore.preprocessor.fix_data`(*cube: Cube, short\_name: str, project: str, dataset: str, mip: str, frequency: str | None = None, session: Session | None = None, \*\*extra\_facets: Any*) → `Cube`

Fix cube data if fixes are required.

This method assumes that metadata is already fixed and checked.

This method collects all the relevant fixes (including generic ones) for a given variable and applies them.

**Parameters**

- **cube** (*Cube*) – Cube to fix.
- **short\_name** (*str*) – Variable's short name.
- **project** (*str*) – Project of the dataset.
- **dataset** (*str*) – Name of the dataset.
- **mip** (*str*) – Variable's MIP.
- **frequency** (*str | None*) – Variable's data frequency, if available.
- **session** (*Session | None*) – Current session which includes configuration and directory information.
- **\*\*extra\_facets** (*Any*) – Extra facets. For details, see *Extra Facets*.

**Returns**

Fixed cube.

**Return type**

`iris.cube.Cube`

`esmvalcore.preprocessor.fix_file`(*file: Path, short\_name: str, project: str, dataset: str, mip: str, output\_dir: Path, add\_unique\_suffix: bool = False, session: Session | None = None, frequency: str | None = None, \*\*extra\_facets: Any*) → `Path | Sequence[Cube]`

Fix files before loading them into a `CubeList`.

This is mainly intended to fix errors that prevent loading the data with Iris (e.g., those related to `missing_value` or `_FillValue`) or operations that are more efficient with other packages (e.g., loading files with lots of variables is much faster with Xarray than Iris).

**Parameters**

- **file** (*Path*) – Path to the original file. Original files are not overwritten.
- **short\_name** (*str*) – Variable's short name.
- **project** (*str*) – Project of the dataset.
- **dataset** (*str*) – Name of the dataset.
- **mip** (*str*) – Variable's MIP.
- **output\_dir** (*Path*) – Output directory for fixed files.

- **add\_unique\_suffix** (*bool*) – Adds a unique suffix to `output_dir` for thread safety.
- **session** (*Session* | *None*) – Current session which includes configuration and directory information.
- **frequency** (*str* | *None*) – Variable's data frequency, if available.
- **\*\*extra\_facets** (*Any*) – Extra facets. For details, see [Extra Facets](#).

**Returns**

Fixed data or a path to them.

**Return type**

Path | Sequence[Cube]

```
esmvalcore.preprocessor.fix_metadata(cubes: Sequence[Cube], short_name: str, project: str, dataset: str,
 mip: str, frequency: str | None = None, session: Session | None =
 None, **extra_facets: Any) → CubeList
```

Fix cube metadata if fixes are required.

This method collects all the relevant fixes (including generic ones) for a given variable and applies them.

**Parameters**

- **cubes** (*Sequence[Cube]*) – Cubes to fix.
- **short\_name** (*str*) – Variable's short name.
- **project** (*str*) – Project of the dataset.
- **dataset** (*str*) – Name of the dataset.
- **mip** (*str*) – Variable's MIP.
- **frequency** (*str* | *None*) – Variable's data frequency, if available.
- **session** (*Session* | *None*) – Current session which includes configuration and directory information.
- **\*\*extra\_facets** (*Any*) – Extra facets. For details, see [Extra Facets](#).

**Returns**

Fixed cubes.

**Return type**

`iris.cube.CubeList`

```
esmvalcore.preprocessor.histogram(cube: Cube, coords: Iterable[Coord] | Iterable[str] | None = None,
 bins: int | Sequence[float] = 10, bin_range: tuple[float, float] | None =
 None, weights: np.ndarray | da.Array | bool | None = None,
 normalization: Literal['sum', 'integral'] | None = None) → Cube
```

Calculate histogram.

Very similar to `numpy.histogram()`, but calculates histogram only over the given coordinates.

Handles lazy data and masked data.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **coords** (*Iterable[Coord]* | *Iterable[str]* | *None*) – Coordinates over which the histogram is calculated. If *None*, calculate the histogram over all coordinates, which results in a scalar cube.

- **bins** (*int* | *Sequence[float]*) – If *bins* is an *int*, it defines the number of equal-width bins in the given *bin\_range*. If *bins* is a sequence, it defines a monotonically increasing array of bin edges, including the rightmost edge, allowing for non-uniform bin widths.
- **bin\_range** (*tuple[float, float]* | *None*) – The lower and upper range of the bins. If *None*, *bin\_range* is simply (`cube.core_data().min()`, `cube.core_data().max()`). Values outside the range are ignored. The first element of the range must be less than or equal to the second. *bin\_range* affects the automatic bin computation as well if *bins* is an *int* (see description for *bins* above).
- **weights** (*np.ndarray* | *da.Array* | *bool* | *None*) – Weights for the histogram calculation. Each value in the input data only contributes its associated weight towards the bin count (instead of 1). Weights are normalized before entering the calculation if *normalization* is 'integral' or 'sum'. Can be an array of the same shape as the input data, *False* or *None* (no weighting), or *True*. In the latter case, weighting will depend on *coords*, and the following coordinates will trigger weighting: *time* (will use lengths of time intervals as weights) and *latitude* (will use cell area weights). Time weights are always calculated from the input data. Area weights can be given as supplementary variables to the recipe (*areacella* or *areacello*, see *Defining supplementary variables (ancillary variables and cell measures)*) or calculated from the input data (this only works for regular grids). By default, **NO** supplementary variables will be used; they need to be explicitly requested in the recipe.
- **normalization** (*Literal['sum', 'integral']* | *None*) – If *None*, the result will contain the number of samples in each bin. If 'integral', the result is the value of the probability *density* function at the bin, normalized such that the integral over the range is 1. If 'sum', the result is the value of the probability *mass* function at the bin, normalized such that the sum over the range is 1. Normalization will be applied across *coords*, not the entire cube.

#### Returns

Histogram cube. The shape of this cube will be (*x1*, *x2*, ..., *n\_bins*), where *xi* are the dimensions of the input cube not appearing in *coords* and *n\_bins* is the number of bins.

#### Return type

Cube

#### Raises

- **TypeError** – Invalid *bin* type given
- **ValueError** – Invalid *normalization* or *bin\_range* given or *bin\_range* is *None* and data is fully masked.
- **iris.exceptions.CoordinateNotFoundError** – A given coordinate of *coords* is not found in cube.
- **iris.exceptions.CoordinateNotFoundError** – *longitude* is not found in cube if *weights=True*, *latitude* is in *coords*, and no *cell\_area* is given as *Defining supplementary variables (ancillary variables and cell measures)*.

`esmvalcore.preprocessor.hourly_statistics(cube: Cube, hours: int, operator: str = 'mean', keep_group_coordinates: bool = False, **operator_kwargs: Any) → Cube`

Compute hourly statistics.

Chunks time in *x* hours periods and computes statistics over them.

#### Parameters

- **cube** (*Cube*) – Input cube.

- **hours** (*int*) – Number of hours per period. Must be a divisor of 24, i.e., (1, 2, 3, 4, 6, 8, 12).
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in *this table*.
- **keep\_group\_coordinates** (*bool*) – If True, keep temporal group coordinates (i.e., `hour_group`, `day_of_year`, `year`) in the result. If False, remove them.
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *operator*.

**Returns**

Hourly statistics cube.

**Return type**

`iris.cube.Cube`

`esmvalcore.preprocessor.linear_trend(cube, coordinate='time')`

Calculate linear trend of data along a given coordinate.

The linear trend is defined as the slope of an ordinary linear regression.

**Parameters**

- **cube** (`iris.cube.Cube`) – Input data.
- **coordinate** (*str, optional (default: 'time')*) – Dimensional coordinate over which the trend is calculated.

**Returns**

Trends.

**Return type**

`iris.cube.Cube`

**Raises**

`iris.exceptions.CoordinateNotFoundError` – The dimensional coordinate with the name `coordinate` is not found in `cube`.

`esmvalcore.preprocessor.linear_trend_stderr(cube, coordinate='time')`

Calculate standard error of linear trend along a given coordinate.

This gives the standard error (not confidence intervals!) of the trend defined as the standard error of the estimated slope of an ordinary linear regression.

**Parameters**

- **cube** (`iris.cube.Cube`) – Input data.
- **coordinate** (*str, optional (default: 'time')*) – Dimensional coordinate over which the standard error of the trend is calculated.

**Returns**

Standard errors of trends.

**Return type**

`iris.cube.Cube`

**Raises**

`iris.exceptions.CoordinateNotFoundError` – The dimensional coordinate with the name `coordinate` is not found in `cube`.

```
esmvalcore.preprocessor.load(file: str | Path | DataElement | Cube | CubeList | Dataset | NcData,
 ignore_warnings: list[dict[str, Any]] | None = None, backend_kwargs:
 dict[str, Any] | None = None) → CubeList
```

Load Iris cubes.

#### Parameters

- **file** (*str* | *Path* | *DataElement* | *Cube* | *CubeList* | *Dataset* | *NcData*) – File to be loaded. If `file` is already a loaded dataset, return it as a `CubeList`. File as `Path` object could be a Zarr store.
- **ignore\_warnings** (*list[dict[str, Any]]* | *None*) – Keyword arguments passed to `warnings.filterwarnings()` used to ignore warnings issued by `iris.load_raw()`. Each list element corresponds to one call to `warnings.filterwarnings()`.
- **backend\_kwargs** (*dict[str, Any]* | *None*) – Dict to hold info needed by storage backend e.g. to access a PRIVATE S3 bucket containing object stores (e.g. netCDF4 files); needed by `fsspec` and its extensions e.g. `s3fs`, so most of the times this will include `storage_options`. Note that Zarr files are opened via `http` extension of `fsspec`, so no need for `storage_options` in that case (ie anon/anon). Currently only used in Zarr file opening.

#### Returns

Loaded cubes.

#### Return type

`iris.cube.CubeList`

#### Raises

- **ValueError** – Cubes are empty.
- **TypeError** – Invalid type for file.

```
esmvalcore.preprocessor.local_solar_time(cube: Cube) → Cube
```

Convert UTC time coordinate to local solar time (LST).

This preprocessor transforms input data with a UTC-based time coordinate to a **local solar time (LST)** coordinate. In LST, 12:00 noon is defined as the moment when the sun reaches its highest point in the sky. Thus, LST is mainly determined by longitude of a location. LST is particularly suited to analyze diurnal cycles across larger regions of the globe, which would be phase-shifted against each other when using UTC time.

To transform data from UTC to LST, this function shifts data along the time axis based on the longitude. In addition to the `cube`'s data, this function also considers auxiliary coordinates, cell measures and ancillary variables that span both the time and longitude dimension.

#### Note

This preprocessor preserves the temporal frequency of the input data. For example, hourly input data will be transformed into hourly output data. For this, a location's exact LST will be put into corresponding bins defined by the bounds of the input time coordinate (in this example, the bin size is 1 hour). If time bounds are not given or cannot be approximated (only one time step is given), a bin size of 1 hour is assumed.

LST is approximated as  $UTC\_time + 12 * longitude / 180$ , where *longitude* is assumed to be in  $[-180, 180]$  (this function automatically calculates the correct format for the longitude). This is only an approximation since the exact LST also depends on the day of year due to the eccentricity of Earth's orbit (see [equation of time](#)). However, since the corresponding error is ~15 min at most, this is ignored here, as most climate model data has a coarser temporal resolution and the time scale for diurnal evolution of meteorological phenomena is usually in the order of hours, not minutes.

**Parameters**

**cube** (*Cube*) – Input cube. Needs a 1D monotonically increasing dimensional coordinate *time* (assumed to refer to UTC time) and a 1D coordinate *longitude*.

**Returns**

Transformed cube of same shape as input cube with an LST coordinate instead of UTC time.

**Return type**

Cube

**Raises**

- **`iris.exceptions.CoordinateNotFoundError`** – Input cube does not contain valid *time* and/or *longitude* coordinate.
- **`iris.exceptions.CoordinateMultiDimError`** – Input cube has multidimensional *longitude* coordinate.
- **`ValueError`** – *time* coordinate of input cube is not monotonically increasing.

`esmvalcore.preprocessor.mask_above_threshold(cube, threshold)`

Mask above a specific threshold value.

Takes a value 'threshold' and masks off anything that is above it in the cube data. Values equal to the threshold are not masked.

**Parameters**

- **cube** (*iris.cube.Cube*) – iris cube to be thresholded.
- **threshold** (*float*) – threshold to be applied on input cube data.

**Returns**

thresholded cube.

**Return type**

*iris.cube.Cube*

`esmvalcore.preprocessor.mask_below_threshold(cube, threshold)`

Mask below a specific threshold value.

Takes a value 'threshold' and masks off anything that is below it in the cube data. Values equal to the threshold are not masked.

**Parameters**

- **cube** (*iris.cube.Cube*) – iris cube to be thresholded
- **threshold** (*float*) – threshold to be applied on input cube data.

**Returns**

thresholded cube.

**Return type**

*iris.cube.Cube*

`esmvalcore.preprocessor.mask_fillvalues(products: Sequence[PreprocessorFile], threshold_fraction: float, min_value: float | None = None, time_window: int = 1) → Sequence[PreprocessorFile]`

Compute and apply a multi-dataset fillvalues mask.

Construct the mask that fills a certain time window with missing values if the number of values in that specific window is less than a given fractional threshold. This function is the extension of `_get_fillvalues_mask` and

performs the combination of missing values masks from each model (of multimodels) into a single fillvalues mask to be applied to each model.

#### Parameters

- **products** (*Sequence[PreprocessorFile]*) – data products to be masked.
- **threshold\_fraction** (*float*) – fractional threshold to be used as argument for Aggregator. Must be between 0 and 1.
- **min\_value** (*float | None*) – minimum value threshold; default None. If default, no thresholding applied so the full mask will be selected.
- **time\_window** (*int*) – time window to compute missing data counts; default set to 1.

#### Returns

Masked iris cubes.

#### Raises

**NotImplementedError** – Implementation missing for data with higher dimensionality than 4.

#### Return type

*Sequence[PreprocessorFile]*

`esmvalcore.preprocessor.mask_glaciated(cube: Cube, mask_out: str = 'glaciated') → Cube`

Mask out glaciated areas.

It applies a Natural Earth mask. Note that for computational reasons only the 10 largest polygons are used for masking.

#### Parameters

- **cube** (*iris.cube.Cube*) – data cube to be masked.
- **mask\_out** (*str*) – “glaciated” to mask out glaciated areas

#### Returns

Returns the masked iris cube.

#### Return type

*iris.cube.Cube*

#### Raises

**ValueError** – Error raised if masking on irregular grids is attempted or if `mask_out` has a wrong value.

`esmvalcore.preprocessor.mask_inside_range(cube, minimum, maximum)`

Mask inside a specific threshold range.

Takes a MINIMUM and a MAXIMUM value for the range, and masks off anything that's between the two in the cube data.

#### Parameters

- **cube** (*iris.cube.Cube*) – iris cube to be thresholded
- **minimum** (*float*) – lower threshold to be applied on input cube data.
- **maximum** (*float*) – upper threshold to be applied on input cube data.

#### Returns

thresholded cube.

#### Return type

*iris.cube.Cube*

`esmvalcore.preprocessor.mask_landsea(cube: Cube, mask_out: Literal['land', 'sea']) → Cube`

Mask out either land mass or sea (oceans, seas and lakes).

It uses dedicated ancillary variables (sftlf or sftof) or, in their absence, it applies a [Natural Earth](#) mask (land or ocean contours). Note that the Natural Earth masks have different resolutions: 10m for land, and 50m for seas. These are more than enough for masking climate model data.

#### Parameters

- **cube** (*Cube*) – Data cube to be masked. If the cube has an `iris.coords.AncillaryVariable` with standard name 'land\_area\_fraction' or 'sea\_area\_fraction' that will be used. If both are present, only the 'land\_area\_fraction' will be used. If the ancillary variable is not available, the mask will be calculated from Natural Earth shapefiles.
- **mask\_out** (*Literal['land', 'sea']*) – Either 'land' to mask out land mass or 'sea' to mask out seas.

#### Returns

Returns the masked iris cube.

#### Return type

`iris.cube.Cube`

#### Raises

**ValueError** – Error raised if masking on irregular grids is attempted without an ancillary variable. Irregular grids are not currently supported for masking with Natural Earth shapefile masks.

`esmvalcore.preprocessor.mask_landseaice(cube: Cube, mask_out: Literal['landsea', 'ice']) → Cube`

Mask out either landsea (combined) or ice.

Function that masks out either landsea (land and seas) or ice (Antarctica, Greenland and some glaciers).

It uses dedicated ancillary variables (sftgif).

#### Parameters

- **cube** (*Cube*) – Data cube to be masked. It should have an `iris.coords.AncillaryVariable` with standard name 'land\_ice\_area\_fraction'.
- **mask\_out** (*str*) – Either 'landsea' to mask out land and oceans or 'ice' to mask out ice.

#### Returns

Returns the masked iris cube with either land or ice masked out.

#### Return type

`iris.cube.Cube`

#### Raises

**ValueError** – Error raised if landsea-ice mask not found as an ancillary variable.

`esmvalcore.preprocessor.mask_multimodel(products)`

Apply common mask to all datasets (using logical OR).

#### Parameters

**products** (*iris.cube.CubeList* or *list of PreprocessorFile*) – Data products/cubes to be masked.

#### Returns

Masked data products/cubes.

**Return type**`iris.cube.CubeList` or list of `PreprocessorFile`**Raises**

- **ValueError** – Datasets have different shapes.
- **TypeError** – Invalid input data.

`esmvalcore.preprocessor.mask_outside_range(cube, minimum, maximum)`

Mask outside a specific threshold range.

Takes a MINIMUM and a MAXIMUM value for the range, and masks off anything that's outside the two in the cube data.

**Parameters**

- **cube** (`iris.cube.Cube`) – iris cube to be thresholded
- **minimum** (`float`) – lower threshold to be applied on input cube data.
- **maximum** (`float`) – upper threshold to be applied on input cube data.

**Returns**

thresholded cube.

**Return type**`iris.cube.Cube`

`esmvalcore.preprocessor.meridional_statistics(cube: Cube, operator: str, normalize: Literal['subtract', 'divide'] | None = None, **operator_kwargs: Any) → Cube`

Compute meridional statistics.

**Parameters**

- **cube** (`Cube`) – Input cube.
- **operator** (`str`) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in [this table](#).
- **normalize** (`Literal['subtract', 'divide'] | None`) – If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be `subtract` (statistics cube is subtracted from the input cube) or `divide` (input cube is divided by the statistics cube).
- **\*\*operator\_kwargs** (`Any`) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by `operator`.

**Returns**

Meridional statistics cube.

**Return type**`iris.cube.Cube`**Raises**

**ValueError** – Error raised if computation on irregular grids is attempted. Zonal statistics not yet implemented for irregular grids.

`esmvalcore.preprocessor.monthly_statistics(cube: Cube, operator: str = 'mean', keep_group_coordinates: bool = False, **operator_kwargs: Any) → Cube`

Compute monthly statistics.

Chunks time in monthly periods and computes statistics over them;

### Parameters

- **cube** (*Cube*) – Input cube.
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in [this table](#).
- **keep\_group\_coordinates** (*bool*) – If True, keep temporal group coordinates (i.e., `month_number`, `year`) in the result. If False, remove them.
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by `operator`.

### Returns

Monthly statistics cube.

### Return type

`iris.cube.Cube`

```
esmvalcore.preprocessor.multi_model_statistics(products: set[PreprocessorFile] | Iterable[Cube],
 span: str, statistics: list[str | dict], output_products:
 dict[str, PreprocessorFile] | None = None, groupby:
 tuple | None = None, keep_input_datasets: bool =
 True, ignore_scalar_coords: bool = False) → dict | set
```

Compute multi-model statistics.

This function computes multi-model statistics on a list of `products`, which can be instances of `Cube` or `PreprocessorFile`. The latter is used internally by `ESMValCore` to store workflow and provenance information, and this option should typically be ignored.

Cubes must have consistent shapes apart from a potential time dimension. There are two options to combine time coordinates of different lengths, see the `span` argument.

Desired statistics need to be given as a list, e.g., `statistics: ['mean', 'median']`. For some statistics like percentiles, it is also possible to pass additional keyword arguments, for example `statistics: [{'operator': 'percentile', 'percent': 20}]`. A full list of supported statistics is available in the section on [Statistical preprocessors](#).

This function can handle cubes with differing metadata:

- Cubes with identical `name()` and `units` will get identical values for `standard_name`, `long_name`, and `var_name` (which will be arbitrarily set to the first encountered value if different cubes have different values for them).
- `attributes`: Differing attributes are deleted, see `iris.util.equalise_attributes()`.
- `cell_methods`: All cell methods are deleted prior to combining cubes.
- `cell_measures()`: All cell measures are deleted prior to combining cubes, see `esmvalcore.preprocessor.remove_fx_variables()`.
- `ancillary_variables()`: All ancillary variables are deleted prior to combining cubes, see `esmvalcore.preprocessor.remove_fx_variables()`.
- `coords()`: Exactly identical coordinates are preserved. For coordinates with equal `name()` and `units`, names are equalized, `attributes` deleted and `circular` is set to False. For all other coordinates, `long_name` is removed, `attributes` deleted and `circular` is set to False. Scalar coordinates can be removed if desired by the option `ignore_scalar_coords=True`. Please note that some special scalar coordinates which are expected to differ across cubes (ancillary coordinates for derived coordinates like `p0` and `ptop`) are always removed.

## Notes

Some of the operators in `iris.analysis` require additional arguments. Except for percentiles, these operators are currently not supported.

Lazy operation is supported for all statistics, except `median`.

### Parameters

- **products** (`set[PreprocessorFile] | Iterable[Cube]`) – Cubes (or products) over which the statistics will be computed.
- **span** (`str`) – Overlap or full; if overlap, statistics are computed on common time- span; if full, statistics are computed on full time spans, ignoring missing data. This option is ignored if input cubes do not have time dimensions.
- **statistics** (`list[str | dict]`) – Statistical operations to be computed, e.g., ['mean', 'median']. For some statistics like percentiles, it is also possible to pass additional keyword arguments, e.g., [{'operator': 'percentile', 'percent': 20}]. All supported options are given in *this table*.
- **output\_products** (`dict[str, PreprocessorFile] | None`) – For internal use only. A dict with statistics names as keys and preprocessorfiles as values. If products are passed as input, the statistics cubes will be assigned to these output products.
- **groupby** (`tuple | None`) – Group products by a given tag or attribute, e.g., ('project', 'dataset', 'tag1'). This is ignored if `products` is a list of cubes.
- **keep\_input\_datasets** (`bool`) – If True, the output will include the input datasets. If False, only the computed statistics will be returned.
- **ignore\_scalar\_coords** (`bool`) – If True, remove any scalar coordinate in the input datasets before merging the input cubes into the multi-dataset cube. The resulting multi-dataset cube will have no scalar coordinates (the actual input datasets will remain unchanged). If False, scalar coordinates will remain in the input datasets, which might lead to merge conflicts in case the input datasets have different scalar coordinates.

### Returns

A `dict` of cubes or `set` of `output_products` depending on the type of `products`.

### Return type

`dict | set`

### Raises

**ValueError** – If `span` is neither overlap nor full, or if input type is neither cubes nor products.

```
esmvalcore.preprocessor.regrid(cube: Cube, target_grid: Cube | Dataset | Path | str | dict, scheme:
 NamedHorizontalScheme | dict, lat_offset: bool = True, lon_offset: bool =
 True, cache_weights: bool = False, use_src_coords: Iterable[str] =
 ('latitude', 'longitude')) → Cube
```

Perform horizontal regridding.

Note that the target grid can be a `Cube`, a `Dataset`, a path to a cube (`Path` or `str`), a grid spec (`str`) in the form of `MxN`, or a `dict` specifying the target grid.

For the latter, the `target_grid` should be a `dict` with the following keys:

- `start_longitude`: longitude at the center of the first grid cell.
- `end_longitude`: longitude at the center of the last grid cell.
- **`step_longitude`: constant longitude distance between grid cell centers.**

- `start_latitude`: latitude at the center of the first grid cell.
- `end_latitude`: longitude at the center of the last grid cell.
- `step_latitude`: constant latitude distance between grid cell centers.

### Parameters

- **cube** (*Cube*) – The source cube to be regridded.
- **target\_grid** (*Cube | Dataset | Path | str | dict*) – The (location of a) cube that specifies the target or reference grid for the regridding operation. Alternatively, a *Dataset* can be provided. Alternatively, a string cell specification may be provided, of the form `MxN`, which specifies the extent of the cell, longitude by latitude (degrees) for a global, regular target grid. Alternatively, a dictionary with a regional target grid may be specified (see above).
- **scheme** (*NamedHorizontalScheme | dict*) – The regridding scheme to perform. If the source grid is structured (i.e., rectilinear or curvilinear), can be one of the built-in schemes `linear`, `nearest`, `area_weighted`. If the source grid is unstructured, can be one of the built-in schemes `linear`, `nearest`. Alternatively, a *dict* that specifies generic regridding can be given (see below).
- **lat\_offset** (*bool*) – Offset the grid centers of the latitude coordinate w.r.t. the pole by half a grid step. This argument is ignored if *target\_grid* is a cube or file.
- **lon\_offset** (*bool*) – Offset the grid centers of the longitude coordinate w.r.t. Greenwich meridian by half a grid step. This argument is ignored if *target\_grid* is a cube or file.
- **cache\_weights** (*bool*) – If `True`, cache regridding weights for later usage. This can speed up the regridding of different datasets with similar source and target grids massively, but may take up a lot of memory for extremely high-resolution data. This option is ignored for schemes that do not support weights caching. More details on this are given in the section on *Reusing regridding weights*. To clear the cache, use `esmvalcore.preprocessor.regrid.cache_clear()`.
- **use\_src\_coords** (*Iterable[str]*) – If there are multiple horizontal coordinates available in the source cube, only use horizontal coordinates with these standard names.

### Returns

Regridded cube.

### Return type

`iris.cube.Cube`

### ➔ See also

#### `extract_levels`

Perform vertical regridding.

### Notes

This preprocessor allows for the use of arbitrary *Iris* regridding schemes, that is anything that can be passed as a scheme to `iris.cube.Cube.regrid()` is possible. This enables the use of further parameters for existing schemes, as well as the use of more advanced schemes for example for unstructured grids. To use this functionality, a dictionary must be passed for the scheme with a mandatory entry of `reference` in the form specified for the object reference of the *entry point data model*, i.e. `importable.module:object.attr`. This is used as a factory for the scheme. Any further entries in the dictionary are passed as keyword arguments to the factory.

For example, to use the familiar `iris.analysis.Linear` regridding scheme with a custom extrapolation mode, use

```
my_preprocessor:
 regrid:
 target: 1x1
 scheme:
 reference: iris.analysis:Linear
 extrapolation_mode: nanmask
```

To use the area weighted regridded available in `esmf_regrid.schemes.ESMFAreaWeighted` use

```
my_preprocessor:
 regrid:
 target: 1x1
 scheme:
 reference: esmf_regrid.schemes:ESMFAreaWeighted
```

`esmvalcore.preprocessor.regrid_time(cube: Cube, frequency: str, calendar: str | None = None, units: str = 'days since 1850-01-01 00:00:00') → Cube`

Align time coordinate for cubes.

Sets datetimes to common values:

- Decadal data (e.g., `frequency='dec'`): 1 January 00:00:00 for the given year. Example: 1 January 2005 00:00:00 for given year 2005 (decade 2000-2010).
- Yearly data (e.g., `frequency='yr'`): 1 July 00:00:00 for each year. Example: 1 July 1993 00:00:00 for the year 1993.
- Monthly data (e.g., `frequency='mon'`): 15th day 00:00:00 for each month. Example: 15 October 1993 00:00:00 for the month October 1993.
- Daily data (e.g., `frequency='day'`): 12:00:00 for each day. Example: 14 March 1996 12:00:00 for the day 14 March 1996.
- $n$ -hourly data where  $n$  is a divisor of 24 (e.g., `frequency='3hr'`): center of each time interval. Example: 03:00:00 for interval 00:00:00-06:00:00 (6-hourly data), 16:30:00 for interval 15:00:00-18:00:00 (3-hourly data), or 09:30:00 for interval 09:00:00-10:00:00 (hourly data).

The corresponding time bounds will be set according to the rules described in `esmvalcore.cmor.fixes.get_time_bounds()`. The data type of the new time coordinate will be set to `float64` (CMOR default for coordinates). Potential auxiliary time coordinates (e.g., `day_of_year`) are also changed if present.

This function does not alter the data in any way.

#### Note

By default, this will not change the calendar of the input time coordinate. For decadal, yearly, and monthly data, it is possible to change the calendar using the `calendar` argument. Be aware that changing the calendar might introduce (small) errors to your data, especially for extensive quantities (those that depend on the period length).

#### Parameters

- **cube** (*Cube*) – Input cube. This input cube will not be modified.
- **frequency** (*str*) – Data frequency. Allowed are

- Decadal data (*frequency* must include *dec*, e.g., *dec*)
- Yearly data (*frequency* must include *yr*, e.g., *yrPt*)
- Monthly data (*frequency* must include *mon*, e.g., *monC*)
- Daily data (*frequency* must include *day*, e.g., *day*)
- *n*-hourly data, where *n* must be a divisor of 24 (*frequency* must include *nh*, e.g., *6hrPt*)
- **calendar** (*str* | *None*) – If given, transform the calendar to the one specified (examples: *standard*, *365\_day*, etc.). This only works for decadal, yearly and monthly data, and will raise an error for other frequencies. If not set, the calendar will not be changed.
- **units** (*str*) – Reference time units used if the calendar of the data is changed. Ignored if *calendar* is not set.

**Returns**

Cube with converted time coordinate.

**Return type**

`iris.cube.Cube`

**Raises**

**NotImplementedError** – An invalid *frequency* is given or *calendar* is set for a non-supported frequency.

`esmvalcore.preprocessor.remove_supplementary_variables(cube: Cube) → Cube`

Remove supplementary variables from cube (in-place).

Strip cell measures or ancillary variables from the cube.

**Parameters**

**cube** (*Cube*) – Iris cube with data and cell measures or ancillary variables.

**Returns**

Cube without cell measures or ancillary variables.

**Return type**

`iris.cube.Cube`

`esmvalcore.preprocessor.resample_hours(cube: Cube, interval: int, offset: int = 0, interpolate: Literal['nearest', 'linear'] | None = None) → Cube`

Convert x-hourly data to y-hourly data.

This is intended to be used with instantaneous data.

**Examples**

- `resample_hours(cube, interval=6)`: Six-hourly intervals at 0:00, 6:00, 12:00, 18:00.
- `resample_hours(cube, interval=6, offset=3)`: Six-hourly intervals at 3:00, 9:00, 15:00, 21:00.
- `resample_hours(cube, interval=12, offset=6)`: Twelve-hourly intervals at 6:00, 18:00.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **interval** (*int*) – The period (hours) of the desired output data.
- **offset** (*int*) – The first hour of the desired output data.

- **interpolate** (*Literal*['nearest', 'linear'] | *None*) – If *interpolate* is *None* (default), convert x-hourly data to y-hourly ( $y > x$ ) by eliminating extra time steps. If *interpolate* is 'nearest' or 'linear', use nearest-neighbor or bilinear interpolation to convert general x-hourly data to general y-hourly data.

**Returns**

Cube with the new frequency.

**Return type**

`iris.cube.Cube`

**Raises**

**ValueError:** – *interval* is not a divisor of 24; invalid *interpolate* given; or input data does not contain any target hour (if *interpolate* is *None*).

`esmvalcore.preprocessor.resample_time(cube: Cube, month: int | None = None, day: int | None = None, hour: int | None = None) → Cube`

Change frequency of data by resampling it.

Converts data from one frequency to another by extracting the timesteps that match the provided month, day and/or hour. This is meant to be used with instantaneous values when computing statistics is not desired.

For example:

- `resample_time(cube, hour=6)`: Daily values taken at 6:00.
- `resample_time(cube, day=15, hour=6)`: Monthly values taken at 15th 6:00.
- `resample_time(cube, month=6)`: Yearly values, taking in June
- `resample_time(cube, month=6, day=1)`: Yearly values, taking 1st June

The condition must yield only one value per interval: the last two samples above will produce yearly data, but the first one is meant to be used to sample from monthly output and the second one will work better with daily.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **month** (*optional*) – Month to extract.
- **day** (*optional*) – Day to extract.
- **hour** (*optional*) – Hour to extract.

**Returns**

Cube with the new frequency.

**Return type**

`iris.cube.Cube`

`esmvalcore.preprocessor.rolling_window_statistics(cube: Cube, coordinate: str, operator: str, window_length: int, **operator_kwargs: Any) → Cube`

Compute rolling-window statistics over a coordinate.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **coordinate** (*str*) – Coordinate over which the rolling-window statistics is calculated.
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in [this table](#).

- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by `operator`.
- **window\_length** (*int*) – Size of the window to use.

**Returns**

Rolling-window statistics cube.

**Return type**

`iris.cube.Cube`

```
esmvalcore.preprocessor.save(cubes: Sequence[Cube], filename: Path | str, optimize_access: str = "",
 compress: bool = False, alias: str = "", compute: bool = True, **kwargs: Any)
 → Delayed | None
```

Save iris cubes to file.

**Parameters**

- **cubes** (*Sequence[Cube]*) – Data cubes to be saved
- **filename** (*Path | str*) – Name of target file
- **optimize\_access** (*str*) – Set internal NetCDF chunking to favour a reading scheme  
Values can be map or timeseries, which improve performance when reading the file one map or time series at a time. Users can also provide a coordinate or a list of coordinates. In that case the better performance will be achieved by loading all the values in that coordinate at a time
- **compress** (*bool*) – Use NetCDF internal compression.
- **alias** (*str*) – Var name to use when saving instead of the one in the cube.
- **compute** (*bool*, *default=True*) – Default is True, meaning complete the file immediately, and return None.

When False, create the output file but don't write any lazy array content to its variables, such as lazy cube data or aux-coord points and bounds. Instead return a `dask.delayed.Delayed` which, when computed, will stream all the lazy content via `dask.store()`, to complete the file. Several such data saves can be performed in parallel, by passing a list of them into a `dask.compute()` call.

- **\*\*kwargs** – See `iris.fileformats.netcdf.saver.save()` for additional keyword arguments.

**Returns**

A delayed object that can be used to save the data in the cube.

**Return type**

`dask.delayed.Delayed` or None

**Raises**

**ValueError** – cubes is empty.

```
esmvalcore.preprocessor.seasonal_statistics(cube: Cube, operator: str = 'mean', seasons: Iterable[str]
 = ('DJF', 'MAM', 'JJA', 'SON'), keep_group_coordinates:
 bool = False, **operator_kwargs: Any) → Cube
```

Compute seasonal statistics.

Chunks time seasons and computes statistics over them.

**Parameters**

- **cube** (*Cube*) – Input cube.

- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Allowed options are given in *this table*.
- **seasons** (*Iterable[str]*) – Seasons to build. Available: ('DJF', 'MAM', 'JJA', 'SON') (default) and all sequentially correct combinations holding every month of a year: e.g. ('JJAS','ONDJFMAM'), or less in case of prior season extraction.
- **keep\_group\_coordinates** (*bool*) – If True, keep temporal group coordinates (i.e., `clim_season`, `season_year`) in the result. If False, remove them.
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *operator*.

**Returns**

Seasonal statistic cube.

**Return type**

`iris.cube.Cube`

```
esmvalcore.preprocessor.timeseries_filter(cube: Cube, window: int, span: int, filter_type: str =
 'lowpass', filter_stats: str = 'sum', **operator_kwargs: Any)
 → Cube
```

Apply a timeseries filter.

Method borrowed from *iris example*

Apply each filter using the `rolling_window` method used with the `weights` keyword argument. A weighted sum is required because the magnitude of the weights are just as important as their relative sizes.

See also the *iris rolling window* `iris.cube.Cube.rolling_window`.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **window** (*int*) – The length of the filter window (in units of cube time coordinate).
- **span** (*int*) – Number of months/days (depending on data frequency) on which weights should be computed e.g. 2-yearly: `span = 24` (2 x 12 months). Span should have same units as cube time coordinate.
- **filter\_type** (*optional*) – Type of filter to be applied; default 'lowpass'. Available types: 'lowpass'.
- **filter\_stats** (*optional*) – Type of statistic to aggregate on the rolling window; default: `sum`. Used to determine the `iris.analysis.Aggregator` object used for aggregation. Allowed options are given in *this table*.
- **\*\*operator\_kwargs** – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *filter\_stats*.

**Returns**

Cube time-filtered using 'rolling\_window'.

**Return type**

`iris.cube.Cube`

**Raises**

- **iris.exceptions.CoordinateNotFoundError**: – Cube does not have time coordinate.
- **NotImplementedError**: – *filter\_type* is not implemented.

`esmvalcore.preprocessor.volume_statistics(cube: Cube, operator: str, normalize: Literal['subtract', 'divide'] | None = None, **operator_kwargs: Any) → Cube`

Apply a statistical operation over a volume.

The volume average is weighted according to the cell volume.

#### Parameters

- **cube** (*Cube*) – Input cube. The input cube should have a `iris.coords.CellMeasure` named 'ocean\_volume', unless it has a `iris.coords.CellMeasure` named 'cell\_area' or regular 1D latitude and longitude coordinates so the cell areas can be computed using `iris.analysis.cartography.area_weights()`. The volume will be computed from the area multiplied by the thickness, computed from the bounds of the vertical coordinate. In that case, vertical coordinate units should be convertible to meters.
- **operator** (*str*) – The operation. Used to determine the `iris.analysis.Aggregator` object used to calculate the statistics. Currently, only *mean* is allowed.
- **normalize** (*Literal['subtract', 'divide'] | None*) – If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be *subtract* (statistics cube is subtracted from the input cube) or *divide* (input cube is divided by the statistics cube).
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the `iris.analysis.Aggregator` object defined by *operator*.

#### Return type

Cube

#### Note

This preprocessor has been designed for oceanic variables, but it might be applicable to atmospheric data as well.

#### Returns

Collapsed cube.

#### Return type

`iris.cube.Cube`

#### Parameters

- **cube** (*Cube*)
- **operator** (*str*)
- **normalize** (*Literal['subtract', 'divide'] | None*)
- **operator\_kwargs** (*Any*)

`esmvalcore.preprocessor.weighting_landsea_fraction(cube, area_type)`

Weight fields using land or sea fraction.

This preprocessor function weights a field with its corresponding land or sea area fraction (value between 0 and 1). The application of this is important for most carbon cycle variables (and other land-surface outputs), which are e.g. reported in units of *kgC m<sup>-2</sup>*. This actually refers to 'per square meter of land/sea' and NOT 'per square meter of gridbox'. So in order to integrate these globally or regionally one has to both area-weight the quantity but also weight by the land/sea fraction.

#### Parameters

- **cube** (*iris.cube.Cube*) – Data cube to be weighted. It should have an *iris.coords.AncillaryVariable* with standard name 'land\_area\_fraction' or 'sea\_area\_fraction'. If both are present, only the 'land\_area\_fraction' will be used.
- **area\_type** (*str*) – Use land ('land') or sea ('sea') fraction for weighting.

**Returns**

Land/sea fraction weighted cube.

**Return type**

*iris.cube.Cube*

**Raises**

- **TypeError** – area\_type is not 'land' or 'sea'.
- **ValueError** – Land/sea fraction variables sftlf or sftof not found.

`esmvalcore.preprocessor.zonal_statistics(cube: Cube, operator: str, normalize: Literal['subtract', 'divide'] | None = None, **operator_kwargs: Any) → Cube`

Compute zonal statistics.

**Parameters**

- **cube** (*Cube*) – Input cube.
- **operator** (*str*) – The operation. Used to determine the *iris.analysis.Aggregator* object used to calculate the statistics. Allowed options are given in *this table*.
- **normalize** (*Literal['subtract', 'divide'] | None*) – If given, do not return the statistics cube itself, but rather, the input cube, normalized with the statistics cube. Can either be *subtract* (statistics cube is subtracted from the input cube) or *divide* (input cube is divided by the statistics cube).
- **\*\*operator\_kwargs** (*Any*) – Optional keyword arguments for the *iris.analysis.Aggregator* object defined by *operator*.

**Returns**

Zonal statistics cube or input cube normalized by statistics cube (see *normalize*).

**Return type**

*iris.cube.Cube*

**Raises**

**ValueError** – Error raised if computation on irregular grids is attempted. Zonal statistics not yet implemented for irregular grids.

## 9.10 Regridding schemes

Iris natively supports data regridding with its *iris.cube.Cube.regrid()* method and a set of predefined regridding schemes provided in the *analysis* module (further details are given on [this page](#)). Here, further regridding schemes are provided that are compatible with *iris.cube.Cube.regrid()*.

Example:

```
from esmvalcore.preprocessor.regrid_schemes import IrisESMFRegrid

regridded_cube = cube.regrid(target_grid, IrisESMFRegrid(method="conservative"))
```

Regridding schemes.

**Classes:**

|                                                              |                                                        |
|--------------------------------------------------------------|--------------------------------------------------------|
| <code>GenericFuncScheme(func, **kwargs)</code>               | Regridding with a generic function.                    |
| <code>GenericRegridder(src_cube, tgt_cube, func, ...)</code> | Generic function regridder.                            |
| <code>IrisESMFRegrid(method[, mdtol, ...])</code>            | <code>iris-esmf-regrid</code> based regridding scheme. |
| <code>UnstructuredLinear()</code>                            | Unstructured bilinear regridding scheme.               |
| <code>UnstructuredLinearRegridder(src_cube, tgt_cube)</code> | Unstructured bilinear regridder.                       |
| <code>UnstructuredNearest()</code>                           | Unstructured nearest-neighbor regridding scheme.       |

**class** `esmvalcore.preprocessor.regrid_schemes.GenericFuncScheme`(*func: Callable, \*\*kwargs: Any*)

Regridding with a generic function.

This class can be used in `iris.cube.Cube.regrid()`.

Does support lazy regridding if *func* does.

**Parameters**

- **func** (*Callable*) – Generic regridding function with signature `f(src_cube: Cube, grid_cube: Cube, **kwargs) -> Cube`.
- **\*\*kwargs** (*Any*) – Keyword arguments for the generic regridding function.

**Methods:**

|                                            |                |
|--------------------------------------------|----------------|
| <code>regridder(src_cube, tgt_cube)</code> | Get regridder. |
|--------------------------------------------|----------------|

**regridder**(*src\_cube: Cube, tgt\_cube: Cube*) → *GenericRegridder*

Get regridder.

**Parameters**

- **src\_cube** (*Cube*) – Cube defining the source grid.
- **tgt\_cube** (*Cube*) – Cube defining the target grid.

**Returns**

Regridder instance.

**Return type**

*GenericRegridder*

**class** `esmvalcore.preprocessor.regrid_schemes.GenericRegridder`(*src\_cube: Cube, tgt\_cube: Cube, func: Callable, \*\*kwargs: Any*)

Generic function regridder.

Does support lazy regridding if *func* does. Does not support weights caching.

**Parameters**

- **src\_cube** (*Cube*) – Cube defining the source grid.
- **tgt\_cube** (*Cube*) – Cube defining the target grid.
- **func** (*Callable*) – Generic regridding function with signature `f(src_cube: Cube, grid_cube: Cube, **kwargs) -> Cube`.
- **\*\*kwargs** (*Any*) – Keyword arguments for the generic regridding function.

```
class esmvalcore.preprocessor.regrid_schemes.IrisESMFRegrid(method: Literal['bilinear',
'conservative', 'nearest'], mdtol: float |
None = None, use_src_mask: None |
bool | np.ndarray = None,
use_tgt_mask: None | bool |
np.ndarray = None,
collapse_src_mask_along:
Iterable[str] = ('Z'),
collapse_tgt_mask_along:
Iterable[str] = ('Z'), src_resolution:
int | None = None, tgt_resolution: int |
None = None, tgt_location:
Literal['face', 'node'] | None = None)
```

`iris-esmf-regrid` based regridding scheme.

This regridding scheme is a thin wrapper around the regridders provided by `esmf_regrid` with improved handling of masks. This allows using these regridders from the `regrid_preprocessor` function.

Supports lazy regridding.

### Parameters

- **method** (*Literal['bilinear', 'conservative', 'nearest']*) – Either “conservative”, “bilinear” or “nearest”. Corresponds to the `esmpy` methods `CONSERVE`, `BILINEAR` or `NEAREST_STOD` used to calculate regridding weights.
- **mdtol** (*float | None*) – Tolerance of missing data. The value returned in each element of the returned array will be masked if the fraction of masked data exceeds `mdtol`. `mdtol=0` means no missing data is tolerated while `mdtol=1` will mean the resulting element will be masked if and only if all the contributing elements of data are masked. If no value is given, this will default to 1 for conservative regridding and 0 otherwise. Only available for methods ‘bilinear’ and ‘conservative’.
- **use\_src\_mask** (*None | bool | np.ndarray*) – If True, derive a mask from the source cube data, which will tell `esmpy` which points to ignore. If an array is provided, that will be used. If set to `None`, it will be set to `True` for methods ‘bilinear’ and ‘conservative’ and to `False` for method ‘nearest’. This default may be changed to `True` for all schemes once [SciTools-incubator/iris-esmf-regrid#368](#) has been resolved.
- **use\_tgt\_mask** (*None | bool | np.ndarray*) – If True, derive a mask from of the target cube, which will tell `esmpy` which points to ignore. If an array is provided, that will be used. If set to `None`, it will be set to `True` for methods ‘bilinear’ and ‘conservative’ and to `False` for method ‘nearest’. This default may be changed to `True` for all schemes once [SciTools-incubator/iris-esmf-regrid#368](#) has been resolved.
- **collapse\_src\_mask\_along** (*Iterable[str]*) – When deriving the mask from the source cube data, collapse the mask along the dimensions identified by these axes or coordinates. Only points that are masked at all time ('T'), vertical levels ('Z'), or both time and vertical levels ('TZ') will be considered masked. Instead of the axes 'T' and 'Z', coordinate names can also be provided. For any cube dimensions not specified here, the first slice along the coordinate will be used to determine the mask.
- **collapse\_tgt\_mask\_along** (*Iterable[str]*) – When deriving the mask from the target cube data, collapse the mask along the dimensions identified by these axes or coordinates. Only points that are masked at all time ('T'), vertical levels ('Z'), or both time and vertical levels ('TZ') will be considered masked. Instead of the axes 'T' and 'Z', coordinate names can also be provided. For any cube dimensions not specified here, the first slice along the coordinate will be used to determine the mask.

- **src\_resolution** (*int* | *None*) – If present, represents the amount of latitude slices per source cell given to ESMF for calculation. If resolution is set, the source cube must have strictly increasing bounds (bounds may be transposed plus or minus 360 degrees to make the bounds strictly increasing). Only available for method 'conservative'.
- **tgt\_resolution** (*int* | *None*) – If present, represents the amount of latitude slices per target cell given to ESMF for calculation. If resolution is set, the target cube must have strictly increasing bounds (bounds may be transposed plus or minus 360 degrees to make the bounds strictly increasing). Only available for method 'conservative'.
- **tgt\_location** (*Literal*['face', 'node'] | *None*) – Only used if the target grid is an `iris.mesh.MeshXY`. Describes the location for data on the mesh. Either 'face' or 'node' for bilinear or nearest neighbour regridding, can only be 'face' for first order conservative regridding.

**kwargs**

Keyword arguments that will be provided to the regridded.

**Methods:**

|                                            |                                                                    |
|--------------------------------------------|--------------------------------------------------------------------|
| <code>regridded(src_cube, tgt_cube)</code> | Create an <code>iris-esmf-regrid</code> based regridding function. |
|--------------------------------------------|--------------------------------------------------------------------|

**regridded**(*src\_cube*: *Cube*, *tgt\_cube*: *Cube* | *MeshXY*) → `ESMFAreaWeightedRegridded` | `ESMFBilinearRegridded` | `ESMFNearestRegridded`

Create an `iris-esmf-regrid` based regridding function.

**Parameters**

- **src\_cube** (*Cube*) – Cube defining the source grid.
- **tgt\_cube** (*Cube* | *MeshXY*) – Cube defining the target grid.

**Returns**

- `esmf_regrid.schemes.ESMFAreaWeightedRegridded` or
- `esmf_regrid.schemes.ESMFBilinearRegridded` or
- `esmf_regrid.schemes.ESMFNearestRegridded` – An `iris-esmf-regrid` regridded.

**Return type**

`ESMFAreaWeightedRegridded` | `ESMFBilinearRegridded` | `ESMFNearestRegridded`

**class** `esmvalcore.preprocessor.regrid_schemes.UnstructuredLinear`

Unstructured bilinear regridding scheme.

This class can be used in `iris.cube.Cube.regrid()`.

Supports lazy regridding.

**⚠ Warning**

This will drop all cell measures, ancillary variables and aux factories, and any auxiliary coordinate that spans the dimension of the unstructured grid.

**Methods:**

|                                            |                |
|--------------------------------------------|----------------|
| <code>regridder(src_cube, tgt_cube)</code> | Get regridder. |
|--------------------------------------------|----------------|

**regridder**(*src\_cube*: *Cube*, *tgt\_cube*: *Cube*) → *UnstructuredLinearRegridder*

Get regridder.

**Parameters**

- **src\_cube** (*Cube*) – Cube defining the source grid.
- **tgt\_cube** (*Cube*) – Cube defining the target grid.

**Returns**

Regridder instance.

**Return type**

*UnstructuredLinearRegridder*

**class** `esmvalcore.preprocessor.regrid_schemes.UnstructuredLinearRegridder`(*src\_cube*: *Cube*,  
*tgt\_cube*: *Cube*)

Unstructured bilinear regridder.

Supports lazy regridding and weights caching.

**Warning**

This will drop all cell measures, ancillary variables and aux factories, and any auxiliary coordinate that spans the dimension of the unstrucuted grid.

**Parameters**

- **src\_cube** (*Cube*) – Cube defining the source grid.
- **tgt\_cube** (*Cube*) – Cube defining the target grid.

**class** `esmvalcore.preprocessor.regrid_schemes.UnstructuredNearest`

Unstructured nearest-neighbor regridding scheme.

This class is a wrapper around `iris.analysis.UnstructuredNearest` that removes any additional X or Y coordinates prior to regridding if necessary. It can be used in `iris.cube.Cube.regrid()`.

**Methods:**

|                                            |                |
|--------------------------------------------|----------------|
| <code>regridder(src_cube, tgt_cube)</code> | Get regridder. |
|--------------------------------------------|----------------|

**regridder**(*src\_cube*: *Cube*, *tgt\_cube*: *Cube*) → *UnstructuredNearestNeighbourRegridder*

Get regridder.

**Parameters**

- **src\_cube** (*Cube*) – Cube defining the source grid.
- **tgt\_cube** (*Cube*) – Cube defining the target grid.

**Returns**

Regridder instance.

**Return type**

*UnstructuredNearestNeighbourRegridder*

## 9.11 Type hints

Type aliases for providing type hints.

```
esmvalcore.typing.DataType = numpy.ndarray | dask.array.core.Array | iris.cube.Cube
```

Type describing data.

```
esmvalcore.typing.FacetValue = str | collections.abc.Sequence[str] | int | float
```

Type describing a single facet.

```
esmvalcore.typing.Facets(*args, **kwargs)
```

Type describing a collection of facets.

alias of `dict[str, str | Sequence[str] | int | float]`

```
esmvalcore.typing.NetCDFAttr = str | int | float | collections.abc.Iterable
```

Type describing netCDF attributes.

NetCDF attributes can be strings, numbers or sequences.

## 9.12 Experimental API

This page describes the new ESMValCore API. The experimental API module is available in the submodule `esmvalcore.experimental`. The API is under development, so use at your own risk!

### 9.12.1 Recipes

This section describes the *recipe* submodule of the API (`esmvalcore.experimental`).

#### Recipe metadata

*Recipe* is a class that holds metadata from a recipe.

```
>>> Recipe('path/to/recipe_python.yml')
recipe = Recipe('Recipe Python')
```

Printing the recipe will give a nice overview of the recipe:

```
>>> print(recipe)
Recipe python

Example recipe that plots a map and timeseries of temperature.

Authors
- Bouwe Andela (NLeSC, Netherlands; https://orcid.org/0000-0001-9005-8940)
- Mattia Righi (DLR, Germany; https://orcid.org/0000-0003-3827-5950)

Maintainers
- Manuel Schlund (DLR, Germany; https://orcid.org/0000-0001-5251-0158)

Projects
- DLR project ESMVal
- Copernicus Climate Change Service 34a Lot 2 (MAGIC) project

References
- Please acknowledge the project(s).
```

## Running a recipe

To run the recipe, call the `run()` method.

```
>>> output = recipe.run()
<log messages>
```

By default, a new Session is automatically created, so that data are never overwritten. Data are stored in the `esmvaltool_output` directory specified in the config. Sessions can also be explicitly specified.

```
>>> from esmvalcore.experimental import CFG
>>> session = CFG.start_session('my_session')
>>> output = recipe.run(session)
<log messages>
```

`run()` returns an dictionary of objects that can be used to inspect the output of the recipe. The output is an instance of `ImageFile` or `DataFile` depending on its type.

For working with recipe output, see: [Recipe output](#).

## Running a single diagnostic or preprocessor task

The python example recipe contains 5 tasks:

Preprocessors:

- `timeseries/tas_amsterdam`
- `timeseries/script1`
- `map/tas`

Diagnostics:

- `timeseries/tas_global`
- `map/script1`

To run a single diagnostic or preprocessor, the name of the task can be passed as an argument to `run()`. If a diagnostic is passed, all ancestors will automatically be run too.

```
>>> output = recipe.run('map/script1')
>>> output
map/script1:
 DataFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.nc')
 DataFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.nc')
 ImageFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.png')
 ImageFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.png')
```

It is also possible to run a single preprocessor task:

```
>>> output = recipe.run('map/tas')
>>> output
map/tas:
 DataFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.nc')
 DataFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.nc')
```

## API reference

Recipe metadata.

### Classes:

|                      |                                               |
|----------------------|-----------------------------------------------|
| <i>Recipe</i> (path) | API wrapper for the esmvalcore Recipe object. |
|----------------------|-----------------------------------------------|

**class** esmvalcore.experimental.recipe.**Recipe**(path: *os.PathLike*)

Bases: `object`

API wrapper for the esmvalcore Recipe object.

This class can be used to inspect and run the recipe.

#### Parameters

**path** (*pathlike*) – Path to the recipe.

#### Attributes:

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>data</i> | Return dictionary representation of the recipe. |
| <i>name</i> | Return the name of the recipe.                  |

#### Methods:

|                              |                         |
|------------------------------|-------------------------|
| <i>get_output</i> ()         | Get output from recipe. |
| <i>render</i> ([template])   | Render output as html.  |
| <i>run</i> ([task, session]) | Run the recipe.         |

**property data:** `dict`

Return dictionary representation of the recipe.

**get\_output**() → *RecipeOutput*

Get output from recipe.

#### Returns

Returns output of the recipe as instances of `OutputFile` grouped by diagnostic task.

#### Return type

output

**property name**

Return the name of the recipe.

**render**(*template=None*)

Render output as html.

#### template

[`Template`] An instance of `jinja2.Template` can be passed to customize the output.

**run**(*task: str | None = None, session: Session | None = None*) → *RecipeOutput*

Run the recipe.

This function loads the recipe into the ESMValCore recipe format and runs it.

#### Parameters

- **task** (*str* / *None*) – Specify the name of the diagnostic or preprocessor to run a single task.
- **session** (*Session* / *None*) – Defines the config parameters and location where the recipe output will be stored. If *None*, a new session will be started automatically.

**Returns**

Returns output of the recipe as instances of `OutputItem` grouped by diagnostic task.

**Return type**

*RecipeOutput*

## 9.12.2 Recipe output

This section describes the `recipe_output` submodule of the API (`esmvalcore.experimental`).

After running a recipe, output is returned by the `run()` method. Alternatively, it can be retrieved using the `get_output()` method.

```
>>> recipe_output = recipe.get_output()
```

`recipe_output` is a mapping of the individual tasks and their output filenames (data and image files) with a set of attributes describing the data.

```
>>> recipe_output
timeseries/script1:
 DataFile('tas_amsterdam_CMIP5_CanESM2_Amon_historical_r1i1p1_tas_1850-2000.nc')
 DataFile('tas_amsterdam_CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_1850-2000.nc')
 DataFile('tas_amsterdam_MultiModelMean_Amon_tas_1850-2000.nc')
 DataFile('tas_global_CMIP5_CanESM2_Amon_historical_r1i1p1_tas_1850-2000.nc')
 DataFile('tas_global_CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_1850-2000.nc')
 ImageFile('tas_amsterdam_CMIP5_CanESM2_Amon_historical_r1i1p1_tas_1850-2000.png')
 ImageFile('tas_amsterdam_CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_1850-2000.png')
 ImageFile('tas_amsterdam_MultiModelMean_Amon_tas_1850-2000.png')
 ImageFile('tas_global_CMIP5_CanESM2_Amon_historical_r1i1p1_tas_1850-2000.png')
 ImageFile('tas_global_CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_1850-2000.png')

map/script1:
 DataFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.nc')
 DataFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.nc')
 ImageFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.png')
 ImageFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.png')
```

Output is grouped by the task that produced them. They can be accessed like a dictionary.

```
>>> task_output = recipe_output['map/script1']
>>> task_output
map/script1:
 DataFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.nc')
 DataFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.nc')
 ImageFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.png')
 ImageFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.png')
```

The task output has a list of files associated with them, usually image (`.png`) or data files (`.nc`). To get a list of all files, use `files()`.

```
>>> print(task_output.files)
(DataFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.nc'),
..., ImageFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.png'))
```

It is also possible to select the image (*image\_files()*) files or data files (*data\_files()*) only.

```
>>> for image_file in task_output.image_files:
>>> print(image_file)
ImageFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.png')
ImageFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.png')

>>> for data_file in task_output.data_files:
>>> print(data_file)
DataFile('CMIP5_CanESM2_Amon_historical_r1i1p1_tas_2000-2000.nc')
DataFile('CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_2000-2000.nc')
```

## Working with output files

Output comes in two kinds, *DataFile* corresponds to data files in .nc format and *ImageFile* corresponds to plots in .png format (see below). Both object are derived from the same base class (*OutputFile*) and therefore share most of the functionality.

For example, author information can be accessed as instances of *Contributor* via

```
>>> output_file = task_output[0]
>>> output_file.authors
(Contributor('Andela, Bouwe', institute='NLeSC, Netherlands', orcid='https://orcid.org/
↪0000-0001-9005-8940'),
 Contributor('Righi, Mattia', institute='DLR, Germany', orcid='https://orcid.org/0000-
↪0003-3827-5950'))
```

And associated references as instances of *Reference* via

```
>>> output_file.references
(Reference('acknow_project'),)
```

*OutputFile* also knows about associated files

```
>>> data_file.citation_file
Path('.../tas_global_CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_1850-2000_citation.
↪bibtex')
>>> data_file.data_citation_file
Path('.../tas_global_CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_1850-2000_data_citation_
↪info.txt')
>>> data_file.provenance_svg_file
Path('.../tas_global_CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_1850-2000_provenance.svg
↪')
>>> data_file.provenance_xml_file
Path('.../tas_global_CMIP6_BCC-ESM1_Amon_historical_r1i1p1f1_tas_1850-2000_provenance.xml
↪')
```

## Working with image files

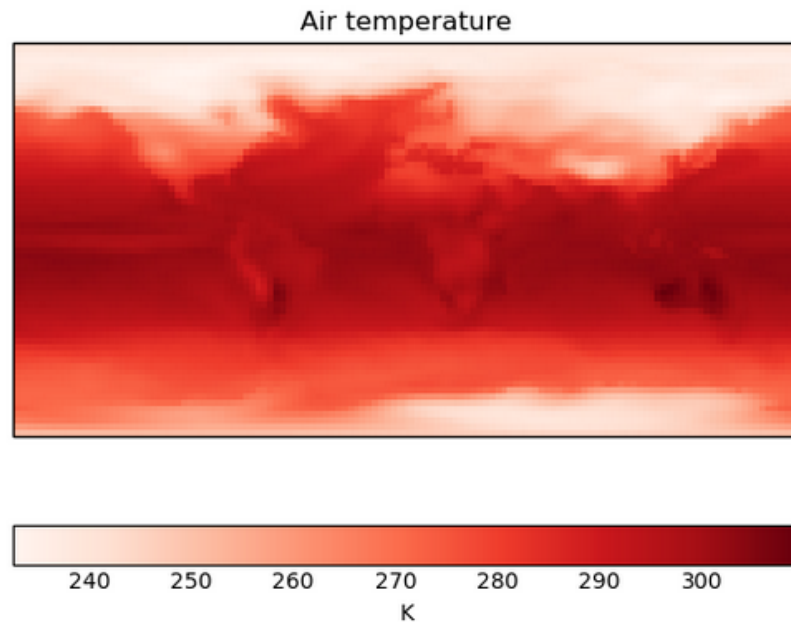
Image output uses IPython magic to plot themselves in a notebook environment.

```
>>> image_file = recipe_output['map/script1'].image_files[0]
>>> image_file
```

For example:

```
In [30]: image_file = recipe_output['map/script1'].image_files[0]
 image_file
```

Out[30]: Average Near-Surface Air Temperature between 2000 and 2000 according to BCC-ESM1.



Using IPython.display, it is possible to show all image files.

```
>>> from IPython.display import display
>>>
>>> task = recipe_output['map/script1']
>>> for image_file in task.image_files:
>>> display(image_file)
```

## Working with data files

Data files can be easily loaded using xarray:

```
>>> data_file = recipe_output['timeseries/script1'].data_files[0]
>>> data = data_file.load_xarray()
>>> type(data)
xarray.core.dataset.Dataset
```

Or iris:

```
>>> cube = data_file.load_iris()
>>> type(cube)
iris.cube.CubeList
```

## API reference

API for handing recipe output.

### Classes:

|                                                                 |                                         |
|-----------------------------------------------------------------|-----------------------------------------|
| <code>DataFile</code> (path[, attributes])                      | Container for data output.              |
| <code>DiagnosticOutput</code> (name, task_output[, title, ...]) | Container for diagnostic output.        |
| <code>ImageFile</code> (path[, attributes])                     | Container for image output.             |
| <code>OutputFile</code> (path[, attributes])                    | Base container for recipe output files. |
| <code>RecipeOutput</code> (task_output, session, info)          | Container for recipe output.            |
| <code>TaskOutput</code> (name, files)                           | Container for task output.              |

**class** `esmvalcore.experimental.recipe_output.DataFile`(*path*: *str*, *attributes*: *dict* | *None* = *None*)

Bases: `OutputFile`

Container for data output.

### Attributes:

|                                  |                                                    |
|----------------------------------|----------------------------------------------------|
| <code>authors</code>             | List of recipe authors.                            |
| <code>caption</code>             | Return the caption of the file (fallback to path). |
| <code>citation_file</code>       | Return path of citation file (bibtex format).      |
| <code>data_citation_file</code>  | Return path of data citation info (txt format).    |
| <code>kind</code>                |                                                    |
| <code>provenance_xml_file</code> | Return path of provenance file (xml format).       |
| <code>references</code>          | List of project references.                        |

### Methods:

|                                          |                                                      |
|------------------------------------------|------------------------------------------------------|
| <code>create</code> (path[, attributes]) | Construct new instances of <code>OutputFile</code> . |
| <code>load_iris</code> ()                | Load data using iris.                                |
| <code>load_xarray</code> ()              | Load data using xarray.                              |

### Parameters

- **path** (*str*)
- **attributes** (*dict* | *None*)

**property authors:** `tuple`

List of recipe authors.

**property caption:** `str`

Return the caption of the file (fallback to path).

**property citation\_file**

Return path of citation file (bibtex format).

**classmethod create**(*path: str, attributes: dict | None = None*) → *OutputFile*

Construct new instances of *OutputFile*.

Chooses a derived class if suitable.

**Parameters**

- **path** (*str*)
- **attributes** (*dict | None*)

**Return type**

*OutputFile*

**property data\_citation\_file**

Return path of data citation info (txt format).

**kind:** *str | None = 'data'*

**load\_iris()**

Load data using iris.

**load\_xarray()**

Load data using xarray.

**property provenance\_xml\_file**

Return path of provenance file (xml format).

**property references:** *tuple*

List of project references.

**class** `esmvalcore.experimental.recipe_output.DiagnosticOutput`(*name, task\_output, title=None, description=None*)

Bases: *object*

Container for diagnostic output.

**Parameters**

- **name** (*str*) – Name of the diagnostic
- **title** (*str*) – Title of the diagnostic
- **description** (*str*) – Description of the diagnostic
- **task\_output** (*list of TaskOutput*) – List of task output.

**class** `esmvalcore.experimental.recipe_output.ImageFile`(*path: str, attributes: dict | None = None*)

Bases: *OutputFile*

Container for image output.

**Attributes:**

|                           |                                                    |
|---------------------------|----------------------------------------------------|
| <i>authors</i>            | List of recipe authors.                            |
| <i>caption</i>            | Return the caption of the file (fallback to path). |
| <i>citation_file</i>      | Return path of citation file (bibtex format).      |
| <i>data_citation_file</i> | Return path of data citation info (txt format).    |

continues on next page

Table 94 – continued from previous page

|                            |                                              |
|----------------------------|----------------------------------------------|
| <i>kind</i>                |                                              |
| <i>provenance_xml_file</i> | Return path of provenance file (xml format). |
| <i>references</i>          | List of project references.                  |

**Methods:**

|                                    |                                                        |
|------------------------------------|--------------------------------------------------------|
| <i>create</i> (path[, attributes]) | Construct new instances of OutputFile.                 |
| <i>to_base64</i> ()                | Encode image as base64 to embed in a Jupyter notebook. |

**Parameters**

- **path** (*str*)
- **attributes** (*dict* | *None*)

**property authors:** *tuple*

List of recipe authors.

**property caption:** *str*

Return the caption of the file (fallback to path).

**property citation\_file**

Return path of citation file (bibtex format).

**classmethod create**(*path: str, attributes: dict | None = None*) → *OutputFile*

Construct new instances of OutputFile.

Chooses a derived class if suitable.

**Parameters**

- **path** (*str*)
- **attributes** (*dict* | *None*)

**Return type**

*OutputFile*

**property data\_citation\_file**

Return path of data citation info (txt format).

**kind:** *str* | *None* = 'image'

**property provenance\_xml\_file**

Return path of provenance file (xml format).

**property references:** *tuple*

List of project references.

**to\_base64**() → *str*

Encode image as base64 to embed in a Jupyter notebook.

**Return type**

*str*

**class** `esmvalcore.experimental.recipe_output.OutputFile`(*path*: *str*, *attributes*: *dict* | *None* = *None*)

Bases: `object`

Base container for recipe output files.

Use `OutputFile.create(path='<path>', attributes=attributes)` to initialize a suitable subclass.

#### Parameters

- **path** (*str*) – Name of output file
- **attributes** (*dict*) – Attributes corresponding to the recipe output

#### Attributes:

|                                  |                                                    |
|----------------------------------|----------------------------------------------------|
| <code>authors</code>             | List of recipe authors.                            |
| <code>caption</code>             | Return the caption of the file (fallback to path). |
| <code>citation_file</code>       | Return path of citation file (bibtex format).      |
| <code>data_citation_file</code>  | Return path of data citation info (txt format).    |
| <code>kind</code>                |                                                    |
| <code>provenance_xml_file</code> | Return path of provenance file (xml format).       |
| <code>references</code>          | List of project references.                        |

#### Methods:

|                                         |                                                      |
|-----------------------------------------|------------------------------------------------------|
| <code>create(path[, attributes])</code> | Construct new instances of <code>OutputFile</code> . |
|-----------------------------------------|------------------------------------------------------|

**property authors:** `tuple`

List of recipe authors.

**property caption:** `str`

Return the caption of the file (fallback to path).

**property citation\_file**

Return path of citation file (bibtex format).

**classmethod create**(*path*: *str*, *attributes*: *dict* | *None* = *None*) → `OutputFile`

Construct new instances of `OutputFile`.

Chooses a derived class if suitable.

#### Parameters

- **path** (*str*)
- **attributes** (*dict* | *None*)

#### Return type

`OutputFile`

**property data\_citation\_file**

Return path of data citation info (txt format).

**kind:** `str` | `None` = `None`

**property provenance\_xml\_file**

Return path of provenance file (xml format).

**property references:** `tuple`

List of project references.

```
class esmvalcore.experimental.recipe_output.RecipeOutput(task_output: dict[str, dict[str, Any]],
 session: esmvalcore.config.Session, info:
 RecipeInfo)
```

Bases: `Mapping`

Container for recipe output.

**Parameters**

- **task\_output** (`dict[str, dict[str, Any]]`) – Dictionary with recipe output grouped by task name. Each task value is a mapping of the filenames with the product attributes.
- **session** (`esmvalcore.config.Session`)
- **info** (`RecipeInfo`)

**diagnostics**

Dictionary with recipe output grouped by diagnostic.

**session**

The session used to run the recipe.

**info**

The recipe used to create the output.

**Attributes:**

`FILTER_ATTRS`

**Methods:**

|                                                     |                                                                                    |
|-----------------------------------------------------|------------------------------------------------------------------------------------|
| <code>from_core_recipe_output(recipe_output)</code> | Construct instance from <code>esmvalcore._recipe.recipe.Recipe.get_output</code> . |
| <code>get(k[,d])</code>                             |                                                                                    |
| <code>items()</code>                                |                                                                                    |
| <code>keys()</code>                                 |                                                                                    |
| <code>read_main_log()</code>                        | Read log file.                                                                     |
| <code>read_main_log_debug()</code>                  | Read debug log file.                                                               |
| <code>render([template])</code>                     | Render output as html.                                                             |
| <code>values()</code>                               |                                                                                    |
| <code>write_html()</code>                           | Write output summary to html document.                                             |

**FILTER\_ATTRS:** `tuple = ('realms', 'plot_type', 'plot_types', 'long_names')`

**classmethod** `from_core_recipe_output(recipe_output: dict) → Self`

Construct instance from `esmvalcore._recipe.recipe.Recipe.get_output`.

The core recipe format is not directly compatible with the API. This constructor converts the raw recipe dict to `RecipeInfo`

**Parameters**

**recipe\_output** (*dict*) – Output from *esmvalcore.\_recipe.recipe.Recipe.get\_output*

**Return type**

*Self*

**get**(*k*[, *d*]) → D[*k*] if *k* in D, else *d*. *d* defaults to None.

**items**() → a set-like object providing a view on D's items

**keys**() → a set-like object providing a view on D's keys

**read\_main\_log**() → *str*

Read log file.

**Return type**

*str*

**read\_main\_log\_debug**() → *str*

Read debug log file.

**Return type**

*str*

**render**(*template=None*)

Render output as html.

**template**

[*Template*] An instance of *jinja2.Template* can be passed to customize the output.

**values**() → an object providing a view on D's values

**write\_html**()

Write output summary to html document.

A html file *index.html* gets written to the session directory.

**class** *esmvalcore.experimental.recipe\_output.TaskOutput*(*name: str, files: dict*)

Bases: *object*

Container for task output.

**Parameters**

- **name** (*str*) – Name of the task
- **files** (*dict*) – Mapping of the filenames with the associated attributes.

**Attributes:**

|                    |                                  |
|--------------------|----------------------------------|
| <i>data_files</i>  | Return a tuple of data objects.  |
| <i>image_files</i> | Return a tuple of image objects. |

**Methods:**

|                                  |                                                              |
|----------------------------------|--------------------------------------------------------------|
| <i>from_task</i> ( <i>task</i> ) | Create an instance of <i>TaskOutput</i> from a <i>Task</i> . |
|----------------------------------|--------------------------------------------------------------|

**property data\_files:** *tuple*

Return a tuple of data objects.

**classmethod** `from_task`(*task*: *esmvalcore.\_task.BaseTask*) → *TaskOutput*

Create an instance of *TaskOutput* from a *Task*.

**Parameters**

**task** (*esmvalcore.\_task.BaseTask*)

**Return type**

*TaskOutput*

**property** `image_files`: **tuple**

Return a tuple of image objects.

### 9.12.3 Recipe Metadata

This section describes the *recipe\_metadata* submodule of the API (*esmvalcore.experimental*).

#### API reference

API for recipe metadata.

#### Classes:

|                                               |                                                                     |
|-----------------------------------------------|---------------------------------------------------------------------|
| <i>Contributor</i> (name, institute[, orcid]) | Contains contributor (author or maintainer) information.            |
| <i>Project</i> (project)                      | Use this class to acknowledge a project associated with the recipe. |
| <i>Reference</i> (filename)                   | Parse reference information from bibtex entries.                    |

#### Exceptions:

|                    |                                   |
|--------------------|-----------------------------------|
| <i>RenderError</i> | Error during rendering of object. |
|--------------------|-----------------------------------|

**class** *esmvalcore.experimental.recipe\_metadata.Contributor*(*name*: *str*, *institute*: *str*, *orcid*: *str* | *None* = *None*)

Bases: *object*

Contains contributor (author or maintainer) information.

**Parameters**

- **name** (*str*) – Name of the author, i.e. 'John Doe'
- **institute** (*str*) – Name of the institute
- **orcid** (*str*, *optional*) – ORCID url

**Methods:**

|                               |                                                             |
|-------------------------------|-------------------------------------------------------------|
| <i>from_dict</i> (attributes) | Return an instance of <i>Contributor</i> from a dictionary. |
| <i>from_tag</i> (tag)         | Return an instance of <i>Contributor</i> from a tag (TAGS). |

**classmethod** `from_dict`(*attributes*)

Return an instance of *Contributor* from a dictionary.

**Parameters**

**attributes** (*dict*) – Dictionary containing name / institute [/ orcid].

**classmethod** `from_tag(tag: str) → Contributor`

Return an instance of Contributor from a tag (TAGS).

**Parameters**

**tag** (*str*) – The contributor tags are defined in the authors section in config-references.yml.

**Return type**

Contributor

**class** `esmvalcore.experimental.recipe_metadata.Project(project: str)`

Bases: `object`

Use this class to acknowledge a project associated with the recipe.

**Parameters**

**project** (*str*) – The project title.

**Methods:**

|                            |                                                  |
|----------------------------|--------------------------------------------------|
| <code>from_tag(tag)</code> | Return an instance of Project from a tag (TAGS). |
|----------------------------|--------------------------------------------------|

**classmethod** `from_tag(tag: str) → Project`

Return an instance of Project from a tag (TAGS).

**Parameters**

**tag** (*str*) – The project tags are defined in config-references.yml.

**Return type**

Project

**class** `esmvalcore.experimental.recipe_metadata.Reference(filename: str)`

Bases: `object`

Parse reference information from bibtex entries.

**Parameters**

**filename** (*str*) – Name of the bibtex file.

**Raises**

**NotImplementedError** – If the bibtex file contains more than 1 entry.

**Methods:**

|                                 |                                                    |
|---------------------------------|----------------------------------------------------|
| <code>from_tag(tag)</code>      | Return an instance of Reference from a bibtex tag. |
| <code>render([renderer])</code> | Render the reference.                              |

**classmethod** `from_tag(tag: str) → Reference`

Return an instance of Reference from a bibtex tag.

**Parameters**

**tag** (*str*) – The bibtex tags resolved as `esmvaltool/references/{tag}.bibtex` or the corresponding directory as defined by the diagnostics path.

**Return type**

Reference

**render**(*renderer: str = 'html'*) → str

Render the reference.

**Parameters**

**renderer** (*str*) – Choose the renderer for the string representation. Must be one of: 'plain-text', 'markdown', 'html', 'latex'

**Returns**

Rendered reference

**Return type**

str

**exception** `esmvalcore.experimental.recipe_metadata.RenderError`

Bases: `BaseException`

Error during rendering of object.

**Methods:**

|                                    |                                               |
|------------------------------------|-----------------------------------------------|
| <code>add_note(note, /)</code>     | Add a note to the exception                   |
| <code>with_traceback(tb, /)</code> | Set self.__traceback__ to tb and return self. |

**Attributes:**

|                   |
|-------------------|
| <code>args</code> |
|-------------------|

**add\_note**(*note, /*)

Add a note to the exception

**args**

**with\_traceback**(*tb, /*)

Set self.\_\_traceback\_\_ to tb and return self.

## 9.12.4 Utils

This section describes the `utils` submodule of the API (`esmvalcore.experimental`).

### Finding recipes

One of the first thing we may want to do, is to simply get one of the recipes available in ESMValTool

If you already know which recipe you want to load, call `get_recipe()`.

```
from esmvalcore.experimental import get_recipe
>>> get_recipe('examples/recipe_python')
Recipe('Recipe python')
```

Call the `get_all_recipes()` function to get a list of all available recipes.

```
>>> from esmvalcore.experimental import get_all_recipes
>>> recipes = get_all_recipes()
>>> recipes
```

(continues on next page)

(continued from previous page)

```
[Recipe('Recipe perfmetrics cmip5 4cds'),
 Recipe('Recipe martin18grl'),
 ...
 Recipe('Recipe wflow'),
 Recipe('Recipe pcrglobwb')]
```

To search for a specific recipe, you can use the `find()` method. This takes a search query that looks through the recipe metadata and returns any matches. The query can be a regex pattern, so you can make it as complex as you like.

```
>>> results = recipes.find('climwip')
[Recipe('Recipe climwip')]
```

The recipes are loaded in a `Recipe` object, which knows about the documentation, authors, project, and related references of the recipe. It resolves all the tags, so that it knows which institute an author belongs to and which references are associated with the recipe.

This means you can search for something like this:

```
>>> recipes.find('Geophysical Research Letters')
[Recipe('Recipe martin18grl'),
 Recipe('Recipe climwip'),
 Recipe('Recipe ecs constraints'),
 Recipe('Recipe ecs scatter'),
 Recipe('Recipe ecs'),
 Recipe('Recipe seaice')]
```

## API reference

ESMValCore utilities.

### Classes:

|                                     |                        |
|-------------------------------------|------------------------|
| <code>RecipeList([iterable])</code> | Container for recipes. |
|-------------------------------------|------------------------|

### Functions:

|                                        |                                         |
|----------------------------------------|-----------------------------------------|
| <code>get_all_recipes([subdir])</code> | Return a list of all available recipes. |
| <code>get_recipe(name)</code>          | Get a recipe by its name.               |

**class** `esmvalcore.experimental.utils.RecipeList` (*iterable=()*, /)

Container for recipes.

### Methods:

|                          |                                                          |
|--------------------------|----------------------------------------------------------|
| <code>find(query)</code> | Search for recipes matching the search query or pattern. |
|--------------------------|----------------------------------------------------------|

**find**(*query: Pattern[str]*) → `RecipeList`

Search for recipes matching the search query or pattern.

Searches in the description, authors and project information fields. All matches are returned.

**Parameters**

**query** (*str*, *Pattern*) – String to search for, e.g. `find_recipes('righi')` will return all matching that author. Can be a *regex* pattern.

**Returns**

List of recipes matching the search query.

**Return type**

*RecipeList*

`esmvalcore.experimental.utils.get_all_recipes(subdir: str | None = None) → list`

Return a list of all available recipes.

**Parameters**

**subdir** (*str*) – Sub-directory of the `DIAGNOSTICS.path` to look for recipes, e.g. `get_all_recipes(subdir='examples')`.

**Returns**

List of available recipes

**Return type**

*RecipeList*

`esmvalcore.experimental.utils.get_recipe(name: os.PathLike | str) → Recipe`

Get a recipe by its name.

The function looks first in the local directory, and second in the repository defined by the diagnostic path. The recipe name can be specified with or without extension. The first match will be returned.

**Parameters**

**name** (*str*, *pathlike*) – Name of the recipe file, i.e. `examples/recipe_python.yml`

**Returns**

Instance of `Recipe` which can be used to inspect and run the recipe.

**Return type**

*Recipe*

**Raises**

**FileNotFoundError** – If the name cannot be resolved to a recipe file.

## CHANGELOG

### 10.1 v2.14.0

#### 10.1.1 Highlights

- Support for intake-esgf and new `esmvaltool config` commands have been added (See *Data sources*)
- A new way to configure the CMOR tables and the possibility to use data that is not described in a CMOR table have been implemented (see *CMOR table configuration*)
- CMIP7 data is now supported (see *Pull request #2935*)

This release includes

#### 10.1.2 Backwards incompatible changes

- Remove support for `~/.esmvaltool/config-user.yml` and `~/.esmvaltool/dask.yml` (*Pull request #2878*) by @bouweandela

##### Upgrade instructions

Move the file `~/.esmvaltool/config-user.yml` to `~/.config/esmvaltool/config-user.yml` and replace `~/.esmvaltool/dask.yml` by the *new way of configuring Dask*.

- Remove deprecated ESMPy regridding schemes (*Pull request #2879*) by @bouweandela

##### Upgrade instructions

In most cases, users do not need to do anything as the *default regridding schemes* have been updated so these schemes are no longer used.

If you were using these schemes from Python or through a *Generic regridding scheme* in your recipe, you will need to update it to use *IrisESMFRegrid* instead.

- Add preliminary CMIP7 support (*Pull request #2935*) by @bouweandela

##### Upgrade instructions

Most users will not be affected by these changes, which were introduced to keep the function signatures easy to read.

1. It is no longer possible to pass `derive` as a positional argument to the `esmvalcore.cmor.table.CMIP6Info.get_variable()` method. Please use `derive=True` or `derive=False` instead of `True` or `False` respectively.
2. Similarly, `frequency` and `check_level` are now keyword only arguments for the functions `cmor_check_metadata()`, `cmor_check_data()`, and `cmor_check()`.
3. The argument `table` to the method `esmvalcore.cmor.table.CustomInfo.get_variable()` has been renamed to `table_name` so the signature of this method matches with the same method on the parent class `esmvalcore.cmor.table.InfoBase.get_variable()`.

- Unify handling of group coordinates in temporal statistics preprocessors (Pull request #2787) by @schlunma

#### **i Upgrade instructions**

This PR changes the default behavior of the preprocessor functions

- `monthly_statistics()` (group coordinates `month_number` and `year`)
- `seasonal_statistics()` (group coordinates `clim_season` and `season_year`)
- `annual_statistics()` (group coordinate `year`)
- `decadal_statistics()` (group coordinate `decade`)

Previously, the returned cubes of these preprocessor functions contained the corresponding group coordinates. Now, those coordinates are not present in the returned cubes anymore.

To restore the old behavior, use `keep_group_coordinates=True`.

- Make CMOR tables configurable through new configuration system and deprecate `config-developer.yml` (Pull request #2946) by @bouweandela

#### **i Upgrade instructions**

Importing `esmvalcore.cmor.table` no longer reads the built-in `config-developer.yml` file and users of the deprecated `CMOR_TABLES` object have to load their configuration by importing `esmvalcore.config` before the `CMOR_TABLES` object is populated.

### 10.1.3 Deprecations

- Move the `esmvalcore.local` and `esmvalcore.esgf` modules into the `esmvalcore.io` module (Pull request #2911) by @bouweandela

#### **i Upgrade instructions**

- Importing `esmvalcore.local` is deprecated and will be removed in v2.16. It can be imported as `esmvalcore.io.local` instead.
- Importing `esmvalcore.esgf` is deprecated and will be removed in v2.16. It can be imported as `esmvalcore.io.esgf` instead.

- Move preprocessor output filename template to new configuration (Pull request #2923) by @bouweandela

**i Upgrade instructions**

If you have defined projects in `config-developer.yml` that are not in the default `config-developer.yml` file shipped with ESMValCore, then you will need to move the value of 'output\_file' to

```
projects:
 <insert project name here>:
 preprocessor_filename_template: <insert output_file value here>
```

- Improve ana4MIPs support in the new configuration (Pull request #2932) by @bouweandela

**i Upgrade instructions**

The name of the project is ana4MIPs, see e.g. [here](#). Using the name ana4mips in all lowercase is deprecated, and support for it will be removed in v2.16.0 of ESMValCore.

- Make CMOR tables configurable through new configuration system and deprecate `config-developer.yml` (Pull request #2946) by @bouweandela

**i Upgrade instructions**

- `config-developer.yml` and the configuration setting `config_developer_file`: upgrade instructions are available [here](#).
- In the module `esmvalcore.cmor.table`,
  - \* the function `read_cmor_tables` which reads the tables based on the deprecated `config-developer` file has been deprecated.
  - \* the `cmor_tables_path`, `default`, and `default_table_prefix` arguments to various CMOR table reader classes and the class `CustomInfo` for reading custom CMOR tables have been deprecated because they are no longer needed with the new configuration format.
  - \* the global variable `CMOR_TABLES` holding the CMOR tables has been deprecated because of [Issue #2954](#).

### 10.1.4 Bug fixes

- Fix a bug where the `mask_fillvalues` preprocessor only recorded provenance for the first result (Pull request #2904) by @bouweandela
- Fixed behavior of `--search_data=quick` and `--search_data=complete` (Pull request #2937) by @schlunma
- Two fixes in preprocessor documentation (Pull request #2966) by @LisaBock
- Do not overwrite provenance information written by `ProvenanceLogger` (Pull request #2987) by @schlunma
- Avoid modifying supplementary variables (Pull request #2992) by @bouweandela
- Avoid issue with deepcopying/pickling `IntakeESGFDatasets` (Pull request #2990) by @bouweandela
- Repair `fix_file` (Pull request #2993) by @bouweandela

### 10.1.5 CMOR standard

- Add CCI lst v3 cmor tables (Pull request #2458) by @morobking
- Add the option to ignore the branding\_suffix for derived CMIP7 and custom variables (Pull request #2982) by @bouweandela
- Avoid hardcoded time coordinate names in CMOR check (Pull request #2981) by @bouweandela

### 10.1.6 Configuration

- Avoid logging the configuration multiple times (Pull request #2869) by @bouweandela
- Add an interface for adding new data sources and add support for intake-esgf as a first example (Pull request #2765) by @bouweandela
- Add BSC data configuration (Pull request #2907) by @sloosvel
- Moved dkrz-era5 data source in DKRZ default config to separate file (Pull request #2996) by @schlunma

### 10.1.7 Computational performance improvements

- Make sure all input arrays passed to `dask.array.apply_gufunc` are Dask arrays (Pull request #2969) by @schlunma

### 10.1.8 Documentation

- Fix indentation and add links in changelog for v2.13 (Pull request #2862) by @bouweandela
- Documentation layout to highlight 2 packages (Pull request #2732) by @flicj191
- Document that we follow SPEC 0 (Pull request #2885) by @bouweandela
- Add missing preprocessor name `rolling_window_statistics` to example. (Pull request #2890) by @katjaweigel
- Fix stable documentation build (Pull request #2893) by @bouweandela
- Move ESMValCore release documentation to How-to guide section (Pull request #2889) by @jlenh
- Add ENES-RI Zenodo community (Pull request #2894) by @bouweandela
- Fix the name and version number on `readthedocs` (Pull request #2926) by @bouweandela
- Add updated release timeline doodle (Pull request #2899) by @jlenh
- Fix tiny doc bug in (legacy) ESGF configuration (Pull request #2945) by @valeriupredo
- Pin sphinx<9 (Pull request #2949) by @valeriupredo
- Add Matomo page view tracking for the documentation (Pull request #2955) by @bouweandela
- Add notes to `climate_statistics` docs about shifting of time coordinate (Pull request #2967) by @schlunma
- Update documentation on derived variables with new CMOR tables configuration (Pull request #2979) by @schlunma
- Update contact info (Pull request #2985) by @axel-lauer
- Update contact email in documentation (Pull request #3008) by @axel-lauer

### 10.1.9 Fixes for datasets

- Fix ERA5 native6 fix to handle single monthly-averaged NetCDF files. (Pull request #2512) by @rbeucher
- Expand ICON extra facets (Pull request #2965) by @schlunma
- Copy fixes for obs4MIPs dataset SSMI RSSv07r00 to RSS-v7 (Pull request #2968) by @bouweandela
- Expand ICON-XPP fix for new model version (Pull request #3014) by @schlunma

### 10.1.10 Installation

- Drop support for Python 3.11 (Pull request #2905) by @bouweandela
- Pin iris <3.14.1 to align with iris-esmf-regrid (Pull request #2925) by @bouweandela
- Update pyproject.toml with new license and license-files settings (Pull request #2924) by @bouweandela
- Support for Python 3.14.0 (Pull request #2850) by @valeriupredoi
- Make the tool compatible with numpy 2.4 (Pull request #2941) by @bouweandela
- Update PyPI publish action (Pull request #2939) by @bouweandela
- Unpin pandas (Pull request #2961) by @valeriupredoi

### 10.1.11 Preprocessor

- Extend preprocessor anomalies (Pull request #2871) by @axel-lauer

### 10.1.12 Automatic testing

- Ignore user configuration in configuration reload test (Pull request #2903) by @bouweandela
- Remove an outdated file that is part of the ESMValTool tests (Pull request #2944) by @bouweandela
- Improve the security of GitHub Action workflows (Pull request #2952) by @bouweandela
- Run s3 zarr tests only if s3 object store is online (Pull request #2958) by @valeriupredoi
- Correction to #2958 - add bucket to S3 url (Pull request #2959) by @valeriupredoi
- Fix CMOR test fails in OSX due to case discrepancy tables -> Tables (Pull request #2975) by @valeriupredoi

### 10.1.13 Variable Derivation

- Add derived variable lapserate (Pull request #2872) by @axel-lauer
- Fix derivation for sea ice extent (siextent) (Pull request #2648) by @axel-lauer

### 10.1.14 Improvements

- Update logger to include esmvaltool logs (Pull request #2886) by @bouweandela
- Enable annotations ruff rule (Pull request #2880) by @bouweandela
- Fix issue with wildcards in timerange after introducing data sources (Pull request #2900) by @bouweandela
- Automatically sort \_\_all\_\_ in Python modules (Pull request #2913) by @bouweandela
- Sort \_\_all\_\_ in Python modules (Pull request #2914) by @bouweandela
- Better debug message when local data is not found within the requested timerange (Pull request #2927) by @bouweandela

- Add float to FacetValue and improve preprocessor filename template error message (Pull request #2931) by @bouweandela
- Removed unused code from Dataset class (Pull request #2963) by @schlunma
- Smarter limit on preprocessor filename length (Pull request #3011) by @bouweandela

## 10.2 v2.13.0

### 10.2.1 Highlights

- Extra facets are now more easily configurable with the new configuration system, as described in *Extra Facets*.
- Preprocessor `esmvalcore.preprocessor.extract_surface_from_atm()` allows to extract the surface values of a 3D atmospheric variable based on surface pressure.
- Force the new configuration system through the environment variable `ESMVALTOOL_CONFIG_DIR` as described *here*.
- A set of ruff rules has been enabled in place of Prospector and related tools and Codacy, modernizing our development setup. Additional information is available in the section on *Code quality* in our contribution guidelines.
- An on-the-fly CMORizer for ICON-XPP is now available.
- *Stale issues and pull requests* will now be closed automatically following the *stale* action workflow.

This release includes

### 10.2.2 Backwards incompatible changes

- Remove deprecated horizontal regridding schemes `unstructured_nearest` and `linear_extrapolate` (Pull request #2743) by @schlunma
  - **`unstructured_nearest`: Please use the scheme `nearest` instead. This is an exact replacement for data on unstructured grids. ESMValCore is now able to determine the most suitable regridding scheme based on the input data.**
  - **`linear_extrapolate`: Please use a generic scheme with reference: `iris.analysis:Linear` and `extrapolation_mode: extrapolate` instead. This is an exact replacement, e.g.:**

```
preprocessors:
 regrid_preprocessor:
 regrid:
 target_grid: 2.5x2.5
 scheme:
 reference: iris.analysis:Linear
 extrapolation_mode: extrapolate
```

- Align custom `lwp` table with CMIP6 version (Pull request #2791) by @schlunma There is no way to restore the old behavior. Downstream code needs to be adapted.

### 10.2.3 Deprecations

- Make extra facets configurable using the new configuration system (Pull request #2747) by @schlunma See *Project-specific configuration*. Please use the configuration option `extra_facets` instead. For example, instead of:

```
content of ~/.esmvaltool/extra_facets/cmip6-mapping.yml
CMIP6-MODEL:
```

(continues on next page)

(continued from previous page)

```
Amon:
 tas:
 extra_facet_key: extra_facet_value
```

use the syntax described under *Extra Facets*:

```
projects:
 CMIP6:
 extra_facets:
 CMIP6-MODEL:
 Amon:
 tas:
 extra_facet_key: extra_facet_value
```

## 10.2.4 CMOR standard

- Fix units and values of tos variable in GISS-E2-1-G-CC esm-piControl (Pull request #2689) by @dhohn
- Add entry for n2os variable (Pull request #2724) by @jlenh
- Added custom CMOR tables for *dpn2o* and *n2oflux* (Pull request #2751) by @schlunma
- Add UKESM1-0-LL exception to fix HadGEM3-GC31-LL 'parent\_time\_units' error (Pull request #2782) by @prosku
- Add custom CMOR table for above-ground biomass (agb) (Pull request #2783) by @axel-lauer

## 10.2.5 Community

- Added stale action (Pull request #2759) by @schlunma

## 10.2.6 Documentation

- Unpin upper sphinx but pin nbsphinx  $\geq 0.9.7$  (Pull request #2685) by @valeriupredo
- Explicitly export LC\_ALL for sphinx builds (Pull request #2708) by @valeriupredo
- Remove reference to psy-plot recipe in ESMValTool (Pull request #2741) by @valeriupredo
- Fix Codecov badge in README (Pull request #2754) by @valeriupredo
- Remove mention of Julia in documentation ahead of discontinued support in v2.13 (Pull request #2819) by @jlenh
- Remove codacy badge (Pull request #2822) by @bouweandela
- Replace nbsphinx with myst-nb and remove pin on ipython (Pull request #2838) by @valeriupredo

## 10.2.7 Fixes for datasets

- Fix for obs4MIPs dataset C3S-GTO-ECV-9-0 (toz) (Pull request #2722) by @axel-lauer
- Add on-the-fly CMORizer for ICON-XPP (Pull request #2659) by @schlunma
- Allow *fix\_file* to return dataset objects (Pull request #2579) by @schlunma
- Fix ocean region coordinate in msftmz dataset (CESM2) (Pull request #1607) by @dhohn

## 10.2.8 Installation

- Pin ipython<9.0 (Pull request #2681) by @valeriupredo
- Set a temporary pin *dask* <2025.4.0 while Iris are looking for a solution (Pull request #2720) by @valeriupredo
- Remove support for python 3.10 (Pull request #2730) by @valeriupredo
- Unpin *dask*, pin *iris*, in light of solved #2716 (*iris* saver not working well with latest *Dask* API) (Pull request #2726) by @valeriupredo
- Harmonize *environment.yml* with *pyproject.yml*: identical dependencies (Pull request #2756) by @valeriupredo
- Install *esmvaltool-sample-data*(==0.0.4) from conda forge not PyPI (Pull request #2795) by @valeriupredo
- Use `pip install --no-deps` to avoid overwriting packages from the conda environment (Pull request #2829) by @bouweandela

## 10.2.9 Iris

- Fix test for upstream *iris* version (Pull request #2752) by @schlunma

## 10.2.10 Preprocessor

- Add unit conversion for air mass (Pull request #2698) by @LisaBock
- Add evaporation flux to special unit conversion (Pull request #2691) by @lukruh
- Add preprocessor to extract surface values from 3D atmospheric variables (Pull request #2641) by @jlenh
- Allow adding supplementary variables from a different project and different dataset (Pull request #2755) by @valeriupredo and (Pull request #2771) by @bouweandela
- Move concatenate preprocessor function to its own module (Pull request #2766) by @bouweandela
- Add preprocessor *align\_metadata* (Pull request #2789) by @schlunma
- Fix for *add\_ancillary\_variable* (#2820) (Pull request #2825) by @jlenh
- Set correct calendar when reading file dates (Pull request #2826) by @bouweandela

## 10.2.11 Automatic testing

- Add support for *codacy-ruff* (Pull request #2683) by @valeriupredo
- Use `--no-deps` in upstream tests to avoid installation problems caused by upper pins (Pull request #2710) by @bouweandela
- Temporary pin *dask* !=2025.4.0 (Pull request #2717) by @valeriupredo
- Explicitly call *mamba* env create with arguments in Circle CI configuration (Pull request #2721) by @valeriupredo
- Avoid reading configuration from disk for every single test (Pull request #2767) by @bouweandela
- Do not assume that no warnings are raised during *test\_dataset\_to\_iris* (Pull request #2773) by @schlunma
- Update CircleCI orbs (Pull request #2784) by @bouweandela
- Zarr support (backend, in *esmvalcore.preprocessor.io.py*) (Pull request #2785) by @valeriupredo
- Update *mamba* and *python* (remove pins and allow for Python 3.13) conda-lock file creation Github Action - via *ESMValTool\_sample\_data* being a conda-forge package (Pull request #2792) by @valeriupredo
- Automatically open pull requests to update GitHub Actions (Pull request #2799) by @bouweandela

- Use finer current version for pypa PyPI upload github action (Pull request #2804) by @valeriupredo
- Change zarr exception test to GroupNotFoundError (Pull request #2823) by @jlenh
- Remove failing test `test_load_zarr_local_not_zarr_file` due to zarr version changes (Pull request #2824) by @jlenh
- Repair gha tests after pip install with no deps and add working reference to ipynb Jupyter notebooks in docs (Pull request #2844) by @valeriupredo

### 10.2.12 Variable Derivation

- Add derivation of moisture flux into atmosphere (Pull request #2697) by @LisaBock

### 10.2.13 Improvements

- Add ocean variable to the ACCESS Live CMORiser (Pull request #2601) by @rbeucher
- Remove accidentally added file (Pull request #2701) by @schlunma
- Use `netCDF4.Dataset` to read start and end date from files (Pull request #2728) by @schlunma
- Allow reading facets from filenames (Pull request #2725) by @schlunma
- Always context managers when handling `netCDF4.Dataset` objects (Pull request #2734) by @schlunma
- Enable more ruff rules (Pull request #2715) by @bouweandela
- Show external warnings only in debug log (Pull request #2733) by @schlunma
- Fix sign of ERA5 rlut and rlutcs (Pull request #2748) by @schlunma
- Using `ESMVALTOOL_CONFIG_DIR` will force the usage of new configuration system and ignore old configuration (Pull request #2736) by @schlunma
- Improve error message if cubes do not overlap in time in `multi_model_statistics` with `span=overlap` (Pull request #2762) by @schlunma
- In modules relevant for variable derivation: cleaned code, doc, and added type hints (Pull request #2772) by @schlunma
- Added `CFG.context/Session.context` (Pull request #2778) by @schlunma
- Use `https://esgf-node.ornl.gov/esgf-1-5-bridge` for ESGF searches by default (Pull request #2781) by @bouweandela
- Pass `chunks={}` to Xarray dataset loader for Zarr stores (Pull request #2794) by @valeriupredo
- Enable ruff rule that checks unused arguments (Pull request #2809) by @bouweandela
- Enable ruff rule to prevent print statements (Pull request #2810) by @bouweandela
- Enable ruff rule that enforces using `items()` to iterate over dict key/value pairs (Pull request #2811) by @bouweandela
- Remove use of prospector and related tools and Codacy (Pull request #2818) by @bouweandela
- Commenting out ORNL ESGF node in `esgf_pyclient` config (Pull request #2806) by @jlenh
- Enable ruff rule that prevents overwriting loop variables (Pull request #2813) by @bouweandela
- Enable ruff rule that checks for commented out code (Pull request #2808) by @bouweandela

## 10.3 v2.12.0

### 10.3.1 Highlights

- Preprocessor `esmvalcore.preprocessor.extract_time()` now allows to extract time blocks in each year by making parameters `start_year` and `end_year` optional.
- A new way of *configuring the tool* has been developed.
- Performance improvements:
  - An iris-esmf-regrid scheme has been added to preprocessor `esmvalcore.preprocessor.regrid()`, which improves the regridding of 2D grids and adds the capability to regrid UGRID meshes out of the box.
  - Data is now saved from one preprocessing task at the time when using the distributed scheduler, in order to avoid running out of memory.
  - A better default `num_workers` has been set when using more than one `max_parallel_tasks` with an unconfigured threaded scheduler.
- An on-the-fly cmorizer for ACCESS native data is now available.

This release includes

### 10.3.2 Backwards incompatible changes

- Make derivation of total column ozone (*toz*) more flexible and add derivation of stratospheric and tropospheric column ozone (Pull request #2509) by @schlunma
  - The units of *toz* have been changed from DU to m to be consistent with the CMIP6 CMOR table. To restore the old behaviour, preprocessor `esmvalcore.preprocessor.convert_units()` can be used to set the units back to DU.
- Remove deprecated CMOR fix/check code (Pull request #2552) by @schlunma
  - CMOR fixes and checks have been clearly separated in v2.10.0, and the old code has now been removed. Use functions `esmvalcore.preprocessors.fix_metadata()`, `esmvalcore.preprocessors.fix_data()`, or `esmvalcore.dataset.Dataset.load()` to fix data. Use functions `esmvalcore.preprocessor.cmor_check_metadata()`, `esmvalcore.preprocessor.cmor_check_data()`, or `esmvalcore.preprocessor.cmor_check()` to check data.
- Remove deprecated statistical operators (Pull request #2553) by @schlunma
  - Old statistical operators that have been deprecated in v2.10.0 have now been removed. Please refer to *Statistical preprocessors* for a detailed description on how to use the operators.
- Save all files in a task at the same time to avoid recomputing intermediate results (Pull request #2522) by @bouwe-andela
  - The signature of the preprocessor function `save()` has changed. The function now accepts a `compute` argument that can be `True`, in which case the return value will be `None` or `False`, in which case the return value will be a `Delayed` object that can be used to compute and save the data of the cube.

### 10.3.3 Deprecations

- Merge configuration object from multiple files (instead of one single file) (Pull request #2448) by @schlunma
  - The single configuration file `config-user.yml` has been deprecated in favour of configuration directories. By default, the directory `~/ .config/esmvaltool` will be considered. To switch to the new format run:

```
mkdir -p ~/.config/esmvaltool && mv ~/.esmvaltool/config-user.yml ~/.
↳config/esmvaltool
```

You can also specify the location of the configuration directory with the `--config_dir` flag. Please refer to [Configuration](#) for a detailed description on how to configure the tool.

- Make Dask configurable in our configuration (Pull request #2616) by @schlunma
  - The old Dask configuration file that needed to be located at `~/.esmvaltool/dask.yml` is now deprecated. Please refer to [Dask configuration](#) for a detailed description on how to configure Dask.

### 10.3.4 Bug fixes

- Concatenate by experiment before concatenating all input files (Pull request #2343) by @dhohn
- Update *CFG* with configuration options given via command line (Pull request #2595) by @schlunma
- Avoid a crash when there is a timeout when shutting down the Dask cluster (Pull request #2580) by @bouweandela
- More reliable datasets to recipe conversion (Pull request #2472) by @bouweandela
- Avoid mutating the input cubes when building the combined cube in preprocessor function `multi_model_statistics` (Pull request #2564) by @bouweandela
- Do not change function argument names when decorator `preserve_float_dtype` is used (Pull request #2645) by @schlunma
- Always write target coordinates to source cube after regridding (Pull request #2673) by @schlunma

### 10.3.5 CMOR standard

- New custom variable for tos uncertainty (Pull request #2470) by @LisaBock
- Add `prc` fix for native6 ERA5 CMORization (Pull request #2550) by @malininae
- Added more variables to EMAC extra facets (Pull request #2617) by @schlunma
- Adding custom tables for ETCCDI indices (Pull request #2442) by @malininae

### 10.3.6 Configuration

- Add public `Config.update_from_dirs()` method (Pull request #2538) by @schlunma
- Do not use `Path` objects as configuration dictionary keys to avoid errors in `dask.config.merge` (Pull request #2578) by @schlunma
- Revise the `user-config.yml` to support updated MO user config requirements (Pull request #2658) by @ehogan

### 10.3.7 Computational performance improvements

- Add an iris-esmf-regrid based regridding scheme (Pull request #2457) by @bouweandela
- Miscellaneous lazy preprocessor improvements (Pull request #2520) by @bouweandela
- Only save data from one preprocessing task at a time with the Distributed scheduler (Pull request #2610) by @bouweandela
- Use better defaults when using `max_parallel_tasks` with an unconfigured threaded scheduler (Pull request #2626) by @bouweandela
- Fix OSX compatibility (Pull request #2636) by @bouweandela

- Make sure that supplementary variables and weights have same chunks as parent cube (Pull request #2637) by @schlunma

### 10.3.8 Documentation

- Ignore autosummary warning in documentation build (Pull request #2480) by @bouweandela
- Fix documentation build and broken link (Pull request #2519) by @bouweandela
- Dark mode compatible logo (Pull request #2532) by @lukruh
- Add a pre-commit badge to README (Pull request #2534) by @valeriupredo
- Retire Mambaforge (Pull request #2556) by @valeriupredo
- Readthedocs configuration: temporary revert to miniconda before miniforge3 becomes available (remove Mambaforge) (Pull request #2562) by @valeriupredo
- Remove Docker build badge in README (Pull request #2565) by @valeriupredo
- Optimize documentation about Earth mover distance in distance\_metric preprocessor (Pull request #2423) by @schlunma
- Update the *esmvaltool* command welcome message (Pull request #2635) by @bouweandela
- Pin sphinx to < 8.2 (Pull request #2671) by @bouweandela

### 10.3.9 Fixes for datasets

- Extra facets added for EMAC to map o3, tro3 and aps (Pull request #2501) by @FranziskaWinterstein
- Fix for CMIP6 AWI-ESM-1-1-LR parent time units (Pull request #2507) by @brittaGrusdt
- Use our own unit conversion function in our fixes (Pull request #2560) by @schlunma
- Missing 2m height coordinate and monotonicity for tasmin in CESM2 and CESM2-WACCM (Pull request #2574) by @Karen-A-Garcia
- Monotonicity fixes for Fgoals (Pull request #2603) by @Karen-A-Garcia
- Expand Amon fix of FIO-ESM-2-0 (CMIP6) (Pull request #2619) by @schlunma
- Added fixes for some 3D atmospheric variables of E3SM-1-1 (CMIP6) (Pull request #2620) by @schlunma
- Correct incorrect time bounds in EMAC data (Pull request #2621) by @FranziskaWinterstein
- Do not copy ICON horizontal grid every time it is used (Pull request #2633) by @schlunma
- Fix *oh* for model: EC-Earth3-AerChem mip: AERMonZ (Pull request #2634) by @valeriupredo
- Update CMIP5 EC-EARTH pr fix (Pull request #2666) by @bouweandela
- Add a fix for differing index coord long names in NorESM2-MM and EC-Earth3-Veg-LR (Pull request #2667) by @bouweandela

### 10.3.10 Installation

- Free esmpy of `>=8.6.0` pin and pin iris-grib >=0.20.0 (Pull request #2542) by @valeriupredo`
- Use pyproject.toml instead of setup.py/setup.cfg (Pull request #2540) by @bouweandela
- Use *miniforge3* for our docker builds instead of *mambaforge* (Pull request #2558) by @valeriupredo
- Support Python 3.13 (Pull request #2566) by @valeriupredo
- Pin dask (Pull request #2654) by @sloosvel

### 10.3.11 Iris

- Set iris.FUTURE flags in one place (Pull request #2622) by @bouweandela
- Silence Iris warnings in `area_statistics` preprocessor function (Pull request #2625) by @bouweandela

### 10.3.12 Preprocessor

- Merge input cubes only once when computing lazy multimodel statistics (Pull request #2518) by @bouweandela
- Make `start_year`, `end_year` in `extract_time` optional to obtain time blocks in each year (Pull request #2490) by @malininae
- Adding hurs (relative humidity) derivation script (Pull request #2397) by @malininae
- Added cumulative sum preprocessor (Pull request #2642) by @schlunma
- Unified ignoring of `iris.warnings.IrisVagueMetadataWarning` in preprocessors (Pull request #2646) by @schlunma
- Raise error if weights are used with unweighted aggregator (Pull request #2640) by @schlunma
- Allow using multi model statistics preprocessor on datasets without `timerange` (Pull request #2644) by @schlunma
- Allow using output from `multi_model_statistics` or `ensemble_statistics` as reference for `bias` or `distance_metric` (Pull request #2652) by @schlunma
- Add option to ignore horizontal coordinates if there are multiple when regridding (Pull request #2672) by @bouweandela

### 10.3.13 Observational and re-analysis dataset support

- Add support for native ERA5 data in GRIB format (Pull request #2178) by @schlunma

### 10.3.14 Automatic testing

- Run a nightly test with the development version of dependencies (Pull request #2478) by @bouweandela
- Use ruff formatter and pre-commit (Pull request #2524) by @bouweandela
- Fix tests if deprecated `~/esmvaltool/config-user.yml` file is available (Pull request #2543) by @schlunma
- Disable upstream tests on commits (Pull request #2548) by @bouweandela
- Disable collecting test coverage by default (Pull request #2456) by @bouweandela
- Enable ruff flake8-bugbear rule (Pull request #2536) by @bouweandela
- Pin mamba in conda lock creation github action (Pull request #2561) by @valeriupredo
- [Numpy2] Support for `numpy==2.0.0` (and pin `iris >=3.11`) (Pull request #2395) by @valeriupredo
- Switch back to Python 3.12 for conda lock file creation due to mamba<2 pin (Pull request #2606) by @valeriupredo
- Always ignore user's configuration when running Dask tests (Pull request #2624) by @schlunma
- Restrict runs of cron Github Actions on forks (Pull request #2649) by @valeriupredo
- Fix test that loads realistic GRIB file (Pull request #2665) by @schlunma
- Fix failing test with Dask 2025.2: ours issue not theirs (Pull request #2663) by @valeriupredo

### 10.3.15 Variable Derivation

- Introduction of the variable `prodlnox` for EMAC (Pull request #2499) by @FranziskaWinterstein

### 10.3.16 Improvements

- On-the-fly cmoriser for ACCESS native data (Pull request #2430) by @rhaegar325
- Fix CFF file (Pull request #2476) by @rbeucher
- Write settings.yml parameters in original order (Pull request #2352) by @enekomartinmartinez
- Fix `access-mapping.yml` `extra_facets` title (Pull request #2485) by @rhaegar325
- Remove ability to log on to ESGF (Pull request #2508) by @bouweandela
- Disable automatic fixes by `pre-commit.ci` (Pull request #2527) by @bouweandela
- Ignore reformatting when viewing `git blame` (Pull request #2539) by @bouweandela
- Enable `ruff` `pydocstyle` linter rule (Pull request #2547) by @bouweandela
- Allows relative paths for diagnostic scripts. (Pull request #2329) by @rbeucher
- Fix 2593 Change log INFO to DEBUG (Pull request #2600) by @rbeucher

## 10.4 v2.11.1

### 10.4.1 Highlights

This is a bugfix release which enables lazy computations in more preprocessors and allows installing the latests version of various dependencies, including Iris (v3.11.0).

This release includes

### 10.4.2 Computational performance improvements

- Optimize functions `mask_landsea()`, `mask_landseaice()` and `calculate_volume()` for lazy input (Pull request #2515) by @schlunma

### 10.4.3 Installation

- Remove support for Python 3.9 (Pull request #2447) by @valeriupredo
- Switch to new iris >= 3.10.0 API (Pull request #2500) by @schlunma
- Pin `dask` to avoid 2024.8.0 - problems with masked fill/missing values (Pull request #2504) by @valeriupredo
- Fix rounding of Pandas datetimes in ICON CMORizer to allow installing latest Pandas version (Pull request #2529) by @valeriupredo

### 10.4.4 Automatic testing

- Fix type hint for new `mypy` version (Pull request #2497) by @schlunma
- Reformat datetime strings be in line with new `isodate==0.7.0` and actual ISO8601 and pin `isodate>=0.7.0` (Pull request #2546) by @valeriupredo

## 10.5 v2.11.0

### 10.5.1 Highlights

- Performance improvements have been made to many preprocessors:
  - Preprocessors `esmvalcore.preprocessor.mask_landsea()`, `esmvalcore.preprocessor.mask_landseaice()`, `esmvalcore.preprocessor.mask_glaciated()`, `esmvalcore.preprocessor.extract_levels()` are now lazy
- Several new preprocessors have been added:
  - `esmvalcore.preprocessor.local_solar_time()`
  - `esmvalcore.preprocessor.distance_metrics()`
  - `esmvalcore.preprocessor.histogram()`
- NEW TREND: First time release manager shout-outs!
  - This is the first ESMValTool release managed by the Met Office! We want to shout this out - and for all future first time release managers to shout-out - to celebrate the growing, thriving ESMValTool community.

This release includes

### 10.5.2 Backwards incompatible changes

- Allow contiguous representation of extracted regions (Pull request #2230) by @rebeccaherman1
  - The preprocessor function `esmvalcore.preprocessor.extract_region()` no longer automatically maps the extracted `iris.cube.Cube` to the 0-360 degrees longitude domain. If you need this behaviour, use `cube.intersection(longitude=(0., 360.))` in your Python code after extracting the region. There is no possibility to restore the previous behaviour from a recipe.
- Use `iris.FUTURE.save_split_attrs = True` to remove iris warning (Pull request #2398) by @schlunma
  - Since v3.8.0, Iris explicitly distinguishes between local and global netCDF attributes. ESMValCore adopted this behavior with v2.11.0. With this change, attributes are written as local attributes by default, unless they already existed as global attributes or belong to a special list of global attributes (in which case attributes are written as global attributes). See `iris.cube.CubeAttrsDict` for details.

### 10.5.3 Deprecations

- Refactor regridding (Pull request #2231) by @schlunma
  - This PR deprecated two regridding schemes, which will be removed with ESMValCore v2.13.0:
    - \* `unstructured_nearest`: Please use the scheme `nearest` instead. This is an exact replacement for data on unstructured grids. ESMValCore is now able to determine the most suitable regridding scheme based on the input data.
    - \* `linear_extrapolate`: Please use a generic scheme with `reference: iris.analysis.Linear` and `extrapolation_mode: extrapolate` instead.
- Allow deprecated regridding scheme `linear_extrapolate` in recipe checks (Pull request #2324) by @schlunma
- Allow deprecated regridding scheme `unstructured_nearest` in recipe checks (Pull request #2336) by @schlunma

### 10.5.4 Bug fixes

- Do not overwrite facets from recipe with CMOR table facets for derived variables (Pull request #2255) by @bouweandela
- Fix error message in variable definition check (Pull request #2313) by @enekomartinmartinez
- Unify dtype handling of preprocessors (Pull request #2393) by @schlunma
- Fix bug in `_rechunk_aux_factory_dependencies` (Pull request #2428) by @ehogan
- Avoid loading entire files into memory when downloading from ESGF (Pull request #2434) by @bouweandela
- Preserve cube attribute global vs local when concatenating (Pull request #2449) by @bouweandela

### 10.5.5 CMOR standard

- Also read default custom CMOR tables if custom location is specified (Pull request #2279) by @schlunma
- Add custom CMOR table for total cloud water (tcw) (Pull request #2277) by @axel-lauer
- Add height for `sfcWindmax` in MPI HighRes models (Pull request #2292) by @maliniinae
- Fixed `positive` attribute in custom `rtnt` table (Pull request #2367) by @schlunma
- Fix `positive` attributes in custom CMOR variables (Pull request #2380) by @schlunma
- Log CMOR check and generic fix output to separate file (Pull request #2361) by @schlunma

### 10.5.6 Computational performance improvements

- More lazy fixes and preprocessing functions (Pull request #2325) by @bouweandela
- Made preprocessors `esmvalcore.preprocessor.mask_landsea()`, `esmvalcore.preprocessor.mask_landseaice()` and `esmvalcore.preprocessor.mask_glaciated()` lazy (Pull request #2268) by @joergbenke
- More lazy `esmvalcore.preprocessor.extract_levels()` preprocessor function (Pull request #2120) by @bouweandela
- Use lazy weights for `esmvalcore.preprocessor.climate_statistics()` and `esmvalcore.preprocessor.axis_statistics()` (Pull request #2346) by @schlunma
- Fixed potential memory leak in `esmvalcore.preprocessor.local_solar_time()` (Pull request #2356) by @schlunma
- Cache regridding weights if possible (Pull request #2344) by @schlunma
- Implement lazy area weights (Pull request #2354) by @schlunma
- Avoid large chunks in `esmvalcore.preprocessor.climate_statistics()` preprocessor function with `period='full'` (Pull request #2404) by @bouweandela
- Load data only once for ESMPy regridders (Pull request #2418) by @bouweandela

### 10.5.7 Documentation

- Use short links in changelog (Pull request #2287) by @bouweandela
- National Computing Infrastructure (NCI), Site specific configuration (Pull request #2281) by @rbeucher
- Update `esmvalcore.preprocessor.multi_model_statistics()` doc with latest changes (new operators, etc.) (Pull request #2321) by @schlunma
- Fix Codacy badge (Pull request #2382) by @bouweandela

- Change 'mean' to 'percentile' in doc strings of preprocessor statistics (Pull request #2327) by @lukruh
- Fixed typo in doc about weighted statistics (Pull request #2387) by @schlunma

### 10.5.8 Fixes for datasets

- Fixing missing height 2m coordinates in GFDL-CM4 and KIOST-ESM (Pull request #2294) by @Karen-A-Garcia
- Added fix for wrong units of c1t for CIESM and FIO-ESM-2-0 (Pull request #2330) by @schlunma
- C mip6 gfdl\_cm4: fix tas height fix to work for concatenated scenarios (Pull request #2332) by @mwjury
- Cordex GERICS REMO2015 lon differences above 10e-4 (Pull request #2334) by @mwjury
- Download ICON grid without locking (Pull request #2359) by @bouweandela
- Added ICON fixes for hfls and hfss (Pull request #2360) by @diegokam
- Added ICON fix for rtnt (Pull request #2366) by @diegokam
- Expanded ICON extra facets (Pull request #2379) by @schlunma
- Add 10m height coordinate to SfcWind GFDL-CM4 instead of 2m height (Pull request #2385) by @Karen-A-Garcia
- Cordex wrf381p: fix tas,tasmax,tasmin height (Pull request #2333) by @mwjury
- Several minor fixes needed for marine BGC data. (Pull request #2110) by @ledm

### 10.5.9 Installation

- Pin pandas yet again avoid new 2.2.1 as well (Pull request #2353) by @valeriupredo
- Update Iris pin to avoid using versions with memory issues (Pull request #2408) by @chrisbillowsMO
- Pin esmpy <8.6.0 (Pull request #2402) by @valeriupredo
- Pin numpy<2.0.0 to avoid pulling 2.0.0rcX (Pull request #2415) by @valeriupredo
- Add support for Python=3.12 (Pull request #2228) by @valeriupredo

### 10.5.10 Preprocessor

- New preprocessor: `esmvalcore.preprocessor.local_solar_time()` (Pull request #2258) by @schlunma
- Read derived variables from other MIP tables (Pull request #2256) by @bouweandela
- Added special unit conversion m -> DU for total column ozone (toz) (Pull request #2270) by @schlunma
- Allow cubes as input for `esmvalcore.preprocessor.bias()` preprocessor (Pull request #2183) by @schlunma
- Add normalization with statistics to many statistics preprocessors (Pull request #2189) by @schlunma
- Adding sfcWind derivation from uas and vas (Pull request #2242) by @malininae
- Update interval check in `resample_hours` (Pull request #2362) by @axel-lauer
- Broadcast properly `cell_measures` when using `esmvalcore.preprocessor.extract_shape()` with `decomposed: True` (Pull request #2348) by @sloosvel
- Compute volume from `cell_area` if available (Pull request #2318) by @enekomartinmartinez
- Do not expand wildcards for datasets of derived variables where not all input variables are available (Pull request #2374) by @schlunma

- Modernize `esmvalcore.preprocessor.regrid_time()` and allow setting a common calendar for decadal, yearly, and monthly data (Pull request #2311) by @schlunma
- Added unstructured linear regridding (Pull request #2350) by @schlunma
- Add preprocessors `esmvalcore.preprocessor.distance_metrics()` and `esmvalcore.preprocessor.histogram()` (Pull request #2299) by @schlunma

### 10.5.11 Automatic testing

- Increase resources for testing installation from conda-forge (Pull request #2297) by @bouweandela
- Pin pandas to avoid broken round function (Pull request #2305) by @schlunma
- Remove team reviewers from conda lock generation workflow in Github Actions (Pull request #2307) by @valeriupredo
- Remove mocking from tests in `tests/unit/preprocessor/_regrid/test_extract_point.py` (Pull request #2193) by @ehogan
- Pin `pytest-mypy` plugin to `>=0.10.3` comply with new `pytest==8` (Pull request #2315) by @valeriupredo
- Fix regridding test for unstructured nearest regridding on OSX (Pull request #2319) by @schlunma
- Fix flaky regrid test by clearing LRU cache after each test (Pull request #2322) by @valeriupredo
- Xfail tests/`integration/cmor/_fixes/test_common.py::test_cl_hybrid_height_coord_fix_metadata` while Iris folk fix behaviour (Pull request #2363) by @valeriupredo
- Update codacy reporter orb to latest version (Pull request #2388) by @valeriupredo
- Add calls to `conda list` in Github Action test workflows to inspect environment (Pull request #2391) by @valeriupredo
- Pin pandas yet again `:panda_face: test_icon` fails again with `pandas=2.2.2` (Pull request #2394) by @valeriupredo
- Fixed units of `cl` test data (necessary since `iris>=3.8.0`) (Pull request #2403) by @schlunma

### 10.5.12 Improvements

- Show files of supplementary variables explicitly in log (Pull request #2303) by @schlunma
- Remove warning about logging in to ESGF (Pull request #2326) by @bouweandela
- Do not read `~/esmvaltool/config-user.yml` if `--config-file` is used (Pull request #2309) by @schlunma
- Support loading ICON grid from ICON rootpath (Pull request #2337) by @schlunma
- Handle warnings about invalid units for `iris>=3.8` (Pull request #2378) by @schlunma
- Added note on how to access `index.html` on remote server (Pull request #2276) by @schlunma
- Remove custom fix for concatenation of aux factories now that bug in `iris` is solved (Pull request #2392) by @schlunma
- Ignored `iris` warnings about global attributes (Pull request #2400) by @schlunma
- Add `native6`, `OBS6` and `RAWOBS` rootpaths to `metoffice config-user.yml` template, and remove temporary dir (Pull request #2432) by @alistairsellar

## 10.6 v2.10.0

### 10.6.1 Highlights

- All statistics preprocessors support the same operators and have a common *documentation*. In addition, arbitrary keyword arguments for the statistical operation can be directly given to the preprocessor.
- The output webpage generated by the tool now looks better and provides methods to select and filter the output.
- Improved computational efficiency:
  - Automatic rechunking between preprocessor steps to keep the *graph size* smaller and the *chunk size* optimal.
  - Reduce the size of the dask graph created by `esmvalcore.preprocessor.anomalies()`.
  - Preprocessors `esmvalcore.preprocessor.mask_above_threshold()`, `esmvalcore.preprocessor.mask_below_threshold()`, `esmvalcore.preprocessor.mask_inside_range()`, `esmvalcore.preprocessor.mask_outside_range()` are now lazy.
  - Lazy coordinates bounds are no longer loaded into memory by the CMOR checks and fixes.

This release includes

### 10.6.2 Backwards incompatible changes

- Remove the deprecated option `use_legacy_supplementaries` (Pull request #2202) by @bouweandela
  - The recommended upgrade procedure is to remove `use_legacy_supplementaries` from `config-user.yml` (if it was there) and remove any mention of `fx_variables` from the recipe. If automatically defining the required supplementary variables does not work, define them in the `variable` or `(additional_) datasets` section as described in *Defining supplementary variables (ancillary variables and cell measures)*.
- Use smarter (units-aware) weights (Pull request #2139) by @schlunma
  - Some preprocessors handle units better. For details, see the pull request.
- Removed deprecated configuration option `offline` (Pull request #2213) by @schlunma
  - In *v2.8.0*, we replaced the old `offline` configuration option. From this version on, it stops working. Please refer to *v2.8.0* for upgrade instructions.
- Fix issue with CORDEX datasets requiring different dataset tags for downloads and fixes (Pull request #2066) by @ljoakim
  - Due to the different facets for CORDEX datasets, there was an inconsistency in the fixing mechanism. This change requires changes to existing recipes that use CORDEX datasets. Please refer to the pull request for detailed update instructions.
- For the following changes, no user change is necessary
  - Remove deprecated way of calling `read_cmor_tables()` (Pull request #2201) by @bouweandela
  - Remove deprecated callback argument from preprocessor load function (Pull request #2207) by @bouweandela
  - Remove deprecated preprocessor function `cleanup` (Pull request #2215) by @bouweandela

### 10.6.3 Deprecations

- Clearly separate fixes and CMOR checks (Pull request #2157) by @schlunma
- Added new operators for statistics preprocessor (e.g., `'percentile'`) and allowed passing additional arguments (Pull request #2191) by @schlunma

- This harmonizes the operators for all statistics preprocessors. From this version, the new names can be used; the old arguments will stop working from version 2.12.0. Please refer to *Statistical preprocessors* for a detailed description.

#### 10.6.4 Bug fixes

- Re-add correctly region-extracted cell measures and ancillary variables after `extract_region` (Pull request #2166) by @valeriupredo, @schlunma
- Fix sorting of datasets
  - Fix sorting of ensemble members in `datasets_to_recipe()` (Pull request #2095) by @bouweandela
  - Fix a problem with sorting datasets that have a mix of facet types (Pull request #2238) by @bouweandela
  - Avoid a crash if dataset has supplementary variables (Pull request #2198) by @bouweandela

#### 10.6.5 CMOR standard

- ERA5 on-the-fly CMORizer: changed sign of variables `evspsbl` and `evspsblpot` (Pull request #2115) by @katjaweigel
- Add `ch4` surface custom cmor table entry (Pull request #2168) by @hb326
- Add CMIP3 institutes names used at NCI (Pull request #2152) by @rbeucher
- Added `get_time_bounds()` and `get_next_month()` to public API (Pull request #2214) by @schlunma
- Improve concatenation checks
  - Relax concatenation checks for `--check_level=relax` and `--check_level=ignore` (Pull request #2144) by @sloosvel
  - Fix `concatenate` preprocessor function (Pull request #2240) by @bouweandela
  - Fix time overlap handling in concatenation (Pull request #2247) by @zklaus

#### 10.6.6 Computational performance improvements

- Make *Minimum, maximum and interval masking* preprocessors lazy (Pull request #2169) by @joergbenke
  - Restored usage of numpy in `_mask_with_shp` (Pull request #2209) by @joergbenke
- Do not realize lazy coordinate bounds in CMOR check (Pull request #2146) by @sloosvel
- Rechunk between preprocessor steps (Pull request #2205) by @bouweandela
- Reduce the size of the dask graph created by the `anomalies` preprocessor function (Pull request #2200) by @bouweandela

#### 10.6.7 Documentation

- Add reference to release v2.9.0 in the changelog (Pull request #2130) by @remi-kazoni
- Add merge instructions to release instructions (Pull request #2131) by @zklaus
- Update `mamba` before building environment during Readthedocs build (Pull request #2149) by @valeriupredo
- Ensure compatible `zstandard` and `zstd` versions for `.conda` support (Pull request #2204) by @zklaus
- Remove outdated documentation (Pull request #2210) by @bouweandela
- Remove meercode badge from README because their API is broken (Pull request #2224) by @valeriupredo
- Correct usage help text of version command (Pull request #2232) by @jfrost-mo

- Add `navigation_with_keys: False` to `html_theme_options` in `Readthedocs conf.py` (Pull request #2245) by @valeriupredo
- Replace square badge with roundy shield for Anaconda sticker in README (Pull request #2233, Pull request #2260) by @valeriupredo

### 10.6.8 Fixes for datasets

- Updated doc about fixes and added type hints to fix functions (Pull request #2160) by @schlunma

### 10.6.9 Installation

- Clean-up how pins are written in conda environment file (Pull request #2125) by @valeriupredo
- Use `importlib.metadata` instead of deprecated `pkg_resources` (Pull request #2096) by @bouweandela
- Pin `shapely` to `>=2.0` (Pull request #2075) by @valeriupredo
- Pin Python to `<3.12` in conda environment (Pull request #2272) by @bouweandela

### 10.6.10 Preprocessor

- Improve preprocessor output sorting code (Pull request #2111) by @bouweandela
- Preprocess datasets in the same order as they are listed in the recipe (Pull request #2103) by @bouweandela

### 10.6.11 Automatic testing

- [Github Actions] Compress all bash shell setters into one default option per workflow (Pull request #2126) by @valeriupredo
- [Github Actions] Fix Monitor Tests Github Action (Pull request #2135) by @valeriupredo
- [condalock] update conda lock file (Pull request #2141) by @valeriupredo
- [Condalock] make sure mamba/conda are at latest version by forcing a pinned mamba install (Pull request #2136) by @valeriupredo
- Update code coverage orbs (Pull request #2206) by @bouweandela
- Revisit the comment-triggered Github Actions test (Pull request #2243) by @valeriupredo
- Remove workflow that runs Github Actions tests from PR comment (Pull request #2244) by @valeriupredo

### 10.6.12 Improvements

- Merge v2.9.x into main (Pull request #2128) by @schlunma
- Fix typo in citation file (Pull request #2182) by @bouweandela
- Cleaned and extended function that extracts datetimes from paths (Pull request #2181) by @schlunma
- Add file encoding (and some read modes) at open file step (Pull request #2219) by @valeriupredo
- Check type of argument passed to `read_cmor_tables()` (Pull request #2217) by @valeriupredo
- Dynamic HTML output for monitoring (Pull request #2062) by @bsolino
- Use PyPI's trusted publishers authentication (Pull request #2269) by @valeriupredo

## 10.7 v2.9.0

### 10.7.1 Highlights

It is now possible to use the [Dask distributed scheduler](#), which can significantly reduce the run-time of recipes. Configuration examples and advice are available in [our documentation](#). More work on improving the computational performance is planned, so please share your experiences, good and bad, with this new feature in [Discussion #1763](#).

This release includes

### 10.7.2 Backwards incompatible changes

- Remove deprecated configuration options (Pull request #2056) by @bouweandela
  - The module `esmvalcore.experimental.config` has been removed. To upgrade, import the module from `esmvalcore.config` instead.
  - The module `esmvalcore._config` has been removed. To upgrade, use `esmvalcore.config` instead.
  - The methods `esmvalcore.config.Session.to_config_user` and `esmvalcore.config.Session.from_config_user` have been removed. To upgrade, use `esmvalcore.config.Session` to access the configuration values directly.

### 10.7.3 Bug fixes

- Respect `ignore_warnings` settings from the project configuration in `config-developer.yml` in `esmvalcore.dataset.Dataset.load()` (Pull request #2046) by @schlunma
- Fixed usage of custom location for custom CMOR tables (Pull request #2052) by @schlunma
- Fix issue with writing `index.html` when *running a recipe* with `--resume-from` (Pull request #2055) by @bouweandela
- Fixed bug in ICON CMORizer that lead to shifted time coordinates (Pull request #2038) by @schlunma
- Include `-` in allowed characters for bibtex references (Pull request #2097) by @alistairsellar
- Do not raise an exception if the requested version of a file is not available for all matching files on ESGF (Pull request #2105) by @bouweandela

### 10.7.4 Computational performance improvements

- Add support for *configuring Dask distributed* (Pull request #2049, Pull request #2122) by @bouweandela
- Make `esmvalcore.preprocessor.extract_levels()` lazy (Pull request #1761) by @bouweandela
- Lazy implementation of `esmvalcore.preprocessor.multi_model_statistics()` and `esmvalcore.preprocessor.ensemble_statistics()` (Pull request #968 and Pull request #2087) by @Peter9192
- Avoid realizing data in preprocessor function `esmvalcore.preprocessor.concatenate()` when cubes overlap (Pull request #2109) by @bouweandela

### 10.7.5 Documentation

- Remove unneeded sphinxcontrib extension (Pull request #2047) by @valeriupredo
- Show ESMValTool logo on PyPI webpage (Pull request #2065) by @valeriupredo
- Fix gitter badge in README (Pull request #2118) by @remi-kazeroni
- Add changelog for v2.9.0 (Pull request #2088 and Pull request #2123) by @bouweandela

## 10.7.6 Fixes for datasets

- Pass the `esmvalcore.config.Session` to fixes (Pull request #1988) by @schlunma
- ICON: Allowed specifying vertical grid information in recipe (Pull request #2067) by @schlunma
- Allow specifying `raw_units` for CESM2, EMAC, and ICON CMORizers (Pull request #2043) by @schlunma
- ICON: allow specifying horizontal grid file in recipe/extra facets (Pull request #2078) by @schlunma
- Fix `tas/tos` CMIP6: FIO, KACE, MIROC, IITM (Pull request #2061) by @pepcos
- Add fix for EC-Earth3-Veg `tos` calendar (Pull request #2100) by @bouweandela
- Correct GISS-E2-1-G `tos` units (Pull request #2099) by @bouweandela

## 10.7.7 Installation

- Drop support for Python 3.8 (Pull request #2053) by @bouweandela
- Add python 3.11 to Github Actions package (conda and PyPI) installation tests (Pull request #2083) by @valeriupredo
- Remove `with_mypy` or `with-mypy` optional tool for prospector (Pull request #2108) by @valeriupredo

## 10.7.8 Preprocessor

- Added `period='hourly'` for `esmvalcore.preprocessor.climate_statistics()` and `esmvalcore.preprocessor.anomalies()` (Pull request #2068) by @schlunma
- Support IPCC AR6 regions in `esmvalcore.preprocessor.extract_shape()` (Pull request #2008) by @schlunma

# 10.8 v2.8.1

## 10.8.1 Highlights

This release adds support for Python 3.11 and includes several bugfixes.

This release includes:

## 10.8.2 Bug fixes

- Pin `numpy` !=1.24.3 (Pull request #2011) by @valeriupredo
- Fix a bug in recording provenance for the `mask_multimodel` preprocessor (Pull request #1984) by @schlunma
- Fix ICON hourly data rounding issues (Pull request #2022) by @BauerJul
- Use the default SSL context when using the `extract_location` preprocessor (Pull request #2023) by @ehogan
- Make time-related CMOR fixes work with time dimensions `time1`, `time2`, `time3` (Pull request #1971) by @schlunma
- Always create a cache directory for storing ICON grid files (Pull request #2030) by @schlunma
- Fixed altitude <-> pressure level conversion for masked arrays in the `extract_levels` preprocessor (Pull request #1999) by @schlunma
- Allowed ignoring of scalar time coordinates in the `multi_model_statistics` preprocessor (Pull request #1961) by @schlunma

### 10.8.3 Fixes for datasets

- Add support for hourly ICON data (Pull request #1990) by @BauerJul
- Fix areacello in BCC-CSM2-MR (Pull request #1993) by @remi-kazeroni

### 10.8.4 Installation

- Add support for Python=3.11 (Pull request #1832) by @valeriupredo
- Modernize conda lock file creation workflow with mamba, Mambaforge etc (Pull request #2027) by @valeriupredo
- Pin *libnetcdf*=4.9.1 (Pull request #2072) by @remi-kazeroni

### 10.8.5 Documentation

- Add changelog for v2.8.1 (Pull request #2079) by @bouweandela

### 10.8.6 Automatic testing

- Use mocked *geopy.geocoders.Nominatim* to avoid *ReadTimeoutError* (Pull request #2005) by @schlunma
- Update pre-commit hooks (Pull request #2020) by @bouweandela

## 10.9 v2.8.0

### 10.9.1 Highlights

- ESMValCore now supports wildcards in recipes and offers improved support for ancillary variables and dataset versioning thanks to contributions by @bouweandela. For details, see *Automatically populating a recipe with all available datasets* and *Defining supplementary variables*.
- Support for CORDEX datasets in a rotated pole coordinate system has been added by @sloosvel.
- Native *ICON* output is now made UGRID-compliant on-the-fly to unlock the use of more sophisticated regridding algorithms, thanks to @schlunma.
- The Python API has been extended with the addition of three modules: *esmvalcore.config*, *esmvalcore.dataset*, and *esmvalcore.local*, all these features courtesy of @bouweandela. For details, see our new example *Example notebooks*.
- The preprocessor *multi\_model\_statistics()* has been extended to support more use-cases thanks to contributions by @schlunma. For details, see *Multi-model statistics*.

This release includes:

### 10.9.2 Backwards incompatible changes

Please read the descriptions of the linked pull requests for detailed upgrade instructions.

- The algorithm for automatically defining the ancillary variables and cell measures has been improved (Pull request #1609) by @bouweandela. If this does not work as expected, more examples of how to adapt your recipes are given [here](#) and in the corresponding sections of the *recipe documentation* and the *preprocessor documentation*.
- Remove deprecated features scheduled for removal in v2.8.0 or earlier (Pull request #1826) by @schlunma. Removed *esmvalcore.iris\_helpers.var\_name\_constraint* (has been deprecated in v2.6.0; please use *iris.NameConstraint* with the keyword argument *var\_name* instead) and the option *always\_use\_ne\_mask*

for `esmvalcore.preprocessor.mask_landsea()` (has been deprecated in v2.5.0; the same behavior can now be achieved by specifying `supplementary_variables`).

- No files will be found if a non-existent version of a dataset is specified (Pull request #1835) by @bouweandela. If a version of a dataset is specified in the recipe, the tool will now search for exactly that version, instead of simply using the latest version. Therefore, it is necessary to make sure that the version number in the directory tree matches with the version number in the recipe to find the files.
- The default filename template for obs4MIPs has been updated to better match filenames used in this project in (Pull request #1866) by @bouweandela. This may cause issues if you are storing all the files for obs4MIPs in a directory with no subdirectories per dataset.

### 10.9.3 Deprecations

Please read the descriptions of the linked pull requests for detailed upgrade instructions.

- Various configuration related options that are now available through `esmvalcore.config` have been deprecated (Pull request #1769) by @bouweandela.
- The `fx_variables` preprocessor argument and related features have been deprecated (Pull request #1609) by @bouweandela. See Pull request #1609#Deprecations for more information.
- Combined `offline` and `always_search_esgf` into a single option `search_esgf` (Pull request #1935) @schlunma. The configuration option/command line argument `offline` has been deprecated in favor of `search_esgf`. The previous `offline: true` is now `search_esgf: never` (the default); the previous `offline: false` is now `search_esgf: when_missing`. More details on how to adapt your workflow regarding these new options are given in Pull request #1935 and the [documentation](#).
- `esmvalcore.preprocessor.cleanup()` has been deprecated (Pull request #1949) @schlunma. Please do not use this anymore in the recipe (it is not necessary).

### 10.9.4 Python API

- Support searching ESGF for a specific version of a file and add `esmvalcore.esgf.ESGFFile.facets` (Pull request #1822) by @bouweandela
- Fix issues with searching for files on ESGF (Pull request #1863) by @bouweandela
- Move the `esmvalcore.experimental.config` module to `esmvalcore.config` (Pull request #1769) by @bouweandela
- Add `esmvalcore.local`, a module to search data on the local filesystem (Pull request ##1835) by @bouweandela
- Add `esmvalcore.dataset` module (Pull request #1877) by @bouweandela

### 10.9.5 Bug fixes

- Import from `esmvalcore.config` in the `esmvalcore.experimental` module (Pull request #1816) by @bouweandela
- Added scalar coords of input cubes to output of `esmpy_regrid` (Pull request #1811) by @schlunma
- Fix severe bug in `esmvalcore.preprocessor.mask_fillvalues()` (Pull request #1823) by @schlunma
- Fix LWP of ICON on-the-fly CMORizer (Pull request #1839) by @schlunma
- Fixed issue in irregular regridding regarding scalar coordinates (Pull request #1845) by @schlunma
- Update product attributes and `metadata.yml` with cube metadata before saving files (Pull request #1837) by @schlunma

- Remove an extra space character from a filename (Pull request #1883) by @bouweandela
- Improve resilience of ESGF search (Pull request #1869) by @bouweandela
- Fix issue with no files found if timerange start/end differs in length (Pull request #1880) by @bouweandela
- Add *driver* and *sub\_experiment* tags to generate dataset aliases (Pull request #1886) by @sloosvel
- Fixed time points of native CESM2 output (Pull request #1772) by @schlunma
- Fix type hints for Python versions < 3.10 (Pull request #1897) by @bouweandela
- Fixed *set\_range\_in\_0\_360* for dask arrays (Pull request #1919) by @schlunma
- Made equalized attributes in concatenated cubes consistent across runs (Pull request #1783) by @schlunma
- Fix issue with reading dates from files (Pull request #1936) by @bouweandela
- Add institute name used on ESGF for CMIP5 CanAM4, CanCM4, and CanESM2 (Pull request #1937) by @bouweandela
- Fix issue where data was not loaded and saved (Pull request #1962) by @bouweandela
- Fix type hints for Python 3.8 (Pull request #1795) by @bouweandela
- Update the institute facet of the CSIRO-Mk3L-1-2 model (Pull request #1966) by @remi-kazoni
- Fixed race condition that may result in errors in `esmvalcore.preprocessor.cleanup()` (Pull request #1949) by @schlunma
- Update notebook so it uses supplementaries instead of ancillaries (Pull request #1945) by @bouweandela

### 10.9.6 Documentation

- Fix anaconda badge in README (Pull request #1759) by @valeriupredo
- Fix mistake in the documentation of `esmvalcore.esgf.find_files` (Pull request #1784) by @bouweandela
- Support linking to “stable” ESMValTool version on readthedocs (Pull request #1608) by @bouweandela
- Updated ICON doc with information on usage of `extract_levels` preprocessor (Pull request #1903) by @schlunma
- Add changelog for latest released version v2.7.1 (Pull request #1905) by @valeriupredo
- Update *preprocessor.rst* due to renaming of NCEP dataset to NCEP-NCAR-R1 (Pull request #1908) by @hb326
- Replace timerange nested lists in docs with overview table (Pull request #1940) by @zklaus
- Updated section “backward compatibility” in *contributing.rst* (Pull request #1918) by @axel-lauer
- Add link to ESMValTool release procedure steps (Pull request #1957) by @remi-kazoni
- Synchronize documentation table of contents with ESMValTool (Pull request #1958) by @bouweandela

### 10.9.7 Improvements

- Support wildcards in the recipe and improve support for ancillary variables and dataset versioning (Pull request #1609) by @bouweandela. More details on how to adapt your recipes are given in the corresponding pull request description and in the corresponding sections of the [recipe documentation](#) and the [preprocessor documentation](#).
- Create a session directory with suffix “-1”, “-2”, etc if it already exists (Pull request #1818) by @bouweandela
- Message for users when they use `esmvaltool` executable from `esmvalcore` only (Pull request #1831) by @valeriupredo
- Order recipe output in `index.html` (Pull request #1899) by @bouweandela
- Improve reading facets from ESGF search results (Pull request #1920) by @bouweandela

### 10.9.8 Fixes for datasets

- Fix rotated coordinate grids and *tas* and *pr* for CORDEX datasets (Pull request #1765) by @sloosvel
- Made ICON output UGRID-compliant (on-the-fly) (Pull request #1664) by @schlunma
- Fix automatic download of ICON grid file and make ICON UGRIDization optional (*default: true*) (Pull request #1922) by @schlunma
- Add siconc fixes for EC-Earth3-Veg and EC-Earth3-Veg-LR models (Pull request #1771) by @egalytska
- Fix siconc in KIOST-ESM (Pull request #1829) by @LisaBock
- Extension of ERA5 CMORizer (variable *cl*) (Pull request #1850) by @axel-lauer
- Add standard variable names for EMAC (Pull request #1853) by @FranziskaWinterstein
- Fix for FGOALS-f3-L *clt* (Pull request #1928) by @LisaBock

### 10.9.9 Installation

- Add all deps to the conda-forge environment and suppress installing and reinstalling deps with pip at readthedocs builds (Pull request #1786) by @valeriupredoi
- Pin netCDF4<1.6.1 (Pull request #1805) by @bouweandela
- Unpin NetCF4 (Pull request #1814) by @valeriupredoi
- Unpin flake8 (Pull request #1820) by @valeriupredoi
- Add iris-esmf-regrid as a dependency (Pull request #1809) by @sloosvel
- Pin esmpy<8.4 (Pull request #1871) by @zklaus
- Update esmpy import for ESMF v8.4.0 (Pull request #1876) by @bouweandela

### 10.9.10 Preprocessor

- Allow *esmvalcore.preprocessor.multi\_model\_statistics()* on cubes with arbitrary dimensions (Pull request #1808) by @schlunma
- Smarter removal of coordinate metadata in *esmvalcore.preprocessor.multi\_model\_statistics()* preprocessor (Pull request #1813) by @schlunma
- Allowed usage of *esmvalcore.preprocessor.multi\_model\_statistics()* on single cubes/products (Pull request #1849) by @schlunma
- Allowed usage of *esmvalcore.preprocessor.multi\_model\_statistics()* on cubes with identical *name()* and *units* (but e.g. different *long\_name*) (Pull request #1921) by @schlunma
- Allowed ignoring scalar coordinates in *esmvalcore.preprocessor.multi\_model\_statistics()* (Pull request #1934) by @schlunma
- Refactored *esmvalcore.preprocessor.regrid()* and removed unnecessary code not needed anymore due to new iris version (Pull request #1898) by @schlunma
- Do not realise coordinates during CMOR check (Pull request #1912) by @sloosvel
- Make *esmvalcore.preprocessor.extract\_volume()* work with closed and mixed intervals and allow nearest value selection (Pull request #1930) by @sloosvel

### 10.9.11 Release

- Changelog for v2.8.0rc1 (Pull request #1952) by @remi-kazeroni
- Increase version number for ESMValCore v2.8.0rc1 (Pull request #1955) by @remi-kazeroni
- Changelog for v2.8.0rc2 (Pull request #1959) by @remi-kazeroni
- Increase version number for ESMValCore v2.8.0rc2 (Pull request #1973) by @remi-kazeroni
- Changelog for v2.8.0 (Pull request #1978) by @remi-kazeroni
- Increase version number for ESMValCore v2.8.0 (Pull request #1983) by @remi-kazeroni

### 10.9.12 Automatic testing

- Set implicit optional to true in *mypy* config to avert side effects and test fails from new mypy version (Pull request #1790) by @valeriupredo
- Remove duplicate *implicit\_optional = True* line in *setup.cfg* (Pull request #1791) by @valeriupredo
- Fix failing test due to missing sample data (Pull request #1797) by @bouweandela
- Remove outdated *cmor\_table* facet from data finder tests (Pull request #1798) by @bouweandela
- Modernize tests for *esmvalcore.preprocessor.save()* (Pull request #1799) by @bouweandela
- No more sequential tests since SegFaults were not noticed anymore (Pull request #1819) by @valeriupredo
- Update pre-commit configuration (Pull request #1821) by @bouweandela
- Updated URL of ICON grid file used for testing (Pull request #1914) by @schlunma

### 10.9.13 Variable Derivation

- Add derivation of sea ice extent (Pull request #1695) by @sloosvel

## 10.10 v2.7.1

### 10.10.1 Highlights

This is a bugfix release where we unpin *cf-units* to allow the latest *iris=3.4.0* to be installed. It also includes an update to the default configuration used when searching the ESGF for files, to account for a recent change of the CEDA ESGF index node hostname. The changelog contains only changes that were made to the `main` branch.

### 10.10.2 Installation

- Set the version number on the development branches to one minor version more than the previous release (Pull request #1854) by @bouweandela
- Unpin *cf-units* (Pull request #1770) by @bouweandela

### 10.10.3 Bug fixes

- Improve error handling if an esgf index node is offline (Pull request #1834) by @bouweandela

### 10.10.4 Automatic testing

- Removed unnecessary test that fails with iris 3.4.0 (Pull request #1846) by @schlunma
- Update CEDA ESGF index node hostname (Pull request #1838) by @valeriupredo

## 10.11 v2.7.0

### 10.11.1 Highlights

- We have a new preprocessor function called 'rolling\_window\_statistics' implemented by @malininae
- We have improved the support for native models, refactored native model fixes by adding common base class *NativeDatasetFix*, changed default DRS for reading native ICON output, and added tests for input/output filenames for **ICON** and **EMAC** on-the-fly CMORizer, all these features courtesy of @schlunma
- Performance of preprocessor functions that use time dimensions has been sped up by **two orders of magnitude** thanks to contributions by @bouweandela

This release includes:

### 10.11.2 Backwards incompatible changes

- Change default DRS for reading native ICON output (Pull request #1705) by @schlunma

### 10.11.3 Bug fixes

- Add support for regions stored as MultiPolygon to extract\_shape preprocessor (Pull request #1670) by @bouweandela
- Fixed type annotations for Python 3.8 (Pull request #1700) by @schlunma
- Core *\_io.concatenate()* may fail due to case when one of the cubes is scalar - this fixes that (Pull request #1715) by @valeriupredo
- Pick up esmvalcore badge instead of esmvaltool one in README (Pull request #1749) by @valeriupredo
- Restore support for scalar cubes to time selection preprocessor functions (Pull request #1750) by @bouweandela
- Fix calculation of precipitation flux in EMAC on-the-fly CMORizer (Pull request #1755) by @schlunma

### 10.11.4 Deprecations

- Remove deprecation warning for regrid schemes already deprecated for v2.7.0 (Pull request #1753) by @valeriupredo

### 10.11.5 Documentation

- Add Met Office Installation Method (Pull request #1692) by @mo-tgeddes
- Add MO-paths to config file (Pull request #1709) by @mo-tgeddes
- Update MO obs4MIPs paths in the user configuration file (Pull request #1734) by @mo-tgeddes
- Update *Making a release* section of the documentation (Pull request #1689) by @sloosvel
- Added changelog for v2.7.0 (Pull request #1746) by @valeriupredo
- update CITATION.cff file with 2.7.0 release info (Pull request #1757) by @valeriupredo

### 10.11.6 Improvements

- New preprocessor function 'rolling\_window\_statistics' (Pull request #1702) by @malininae
- Remove *pytest\_flake8* plugin and use *flake8* instead (Pull request #1722) by @valeriupredo
- Added CESM2 CMORizer (Pull request #1678) by @schlunma
- Speed up functions that use time dimension (Pull request #1713) by @bouweandela
- Modernize and minimize pylint configuration (Pull request #1726) by @bouweandela

### 10.11.7 Fixes for datasets

- Refactored native model fixes by adding common base class *NativeDatasetFix* (Pull request #1694) by @schlunma

### 10.11.8 Installation

- Pin *netCDF4* != 1.6.1 since that seems to throw a flurry of Segmentation Faults (Pull request #1724) by @valeriupredo

### 10.11.9 Automatic testing

- Pin *flake8* < 5.0.0 since Circle CI tests are failing copiously (Pull request #1698) by @valeriupredo
- Added tests for input/output filenames for ICON and EMAC on-the-fly CMORizer (Pull request #1718) by @schlunma
- Fix failed tests for Python < 3.10 resulting from typing (Pull request #1748) by @schlunma

## 10.12 v2.6.0

### 10.12.1 Highlights

- A new set of CMOR fixes is now available in order to load native EMAC model output and CMORize it on the fly. For details, see *Supported native models: EMAC*.
- The version number of ESMValCore is now automatically generated using *setuptools\_scm*, which extracts Python package versions from git metadata.

This release includes

### 10.12.2 Deprecations

- Deprecate the function *esmvalcore.var\_name\_constraint* (Pull request #1592) by @schlunma. This function is scheduled for removal in v2.8.0. Please use *iris.NameConstraint* with the keyword argument *var\_name* instead: this is an exact replacement.

### 10.12.3 Bug fixes

- Added *start\_year* and *end\_year* attributes to derived variables (Pull request #1547) by @schlunma
- Show all results on recipe results webpage (Pull request #1560) by @bouweandela
- Regridding regular grids with similar coordinates (Pull request #1567) by @tomaslovato
- Fix timerange wildcard search when deriving variables or downloading files (Pull request #1562) by @sloosvel
- Fix *force\_derivation* bug (Pull request #1627) by @sloosvel

- Correct *build-and-deploy-on-pypi* action (Pull request #1634) by @sloosvel
- Apply *clip\_timerange* to time dependent fx variables (Pull request #1603) by @sloosvel
- Correctly handle requests.exceptions.ConnectTimeout when an ESGF index node is offline (Pull request #1638) by @bouweandela

#### 10.12.4 CMOR standard

- Added custom CMOR tables used for EMAC CMORizer (Pull request #1599) by @schlunma
- Extended ICON CMORizer (Pull request #1549) by @schlunma
- Add CMOR check exception for a basin coord named sector (Pull request #1612) by @dhohn
- Custom user-defined location for custom CMOR tables (Pull request #1625) by @schlunma

#### 10.12.5 Containerization

- Remove update command in Dockerfile (Pull request #1630) by @sloosvel

#### 10.12.6 Community

- Add David Hohn to contributors' list (Pull request #1586) by @valeriupredo

#### 10.12.7 Documentation

- [Github Actions Docs] Full explanation on how to use the GA test triggered by PR comment and added docs link for GA hosted runners (Pull request #1553) by @valeriupredo
- Update the command for building the documentation (Pull request #1556) by @bouweandela
- Update documentation on running the tool (Pull request #1400) by @bouweandela
- Add support for DKRZ-Levante (Pull request #1558) by @remi-kazoni
- Improved documentation on native dataset support (Pull request #1559) by @schlunma
- Tweak *extract\_point* preprocessor: explain what it returns if one point coord outside cube and add explicit test (Pull request #1584) by @valeriupredo
- Update CircleCI, readthedocs, and Docker configuration (Pull request #1588) by @bouweandela
- Remove support for Mistral in *config-user.yml* (Pull request #1620) by @remi-kazoni
- Add changelog for v2.6.0rc1 (Pull request #1633) by @sloosvel
- Add a note on transferring permissions to the release manager (Pull request #1645) by @bouweandela
- Add documentation on building and uploading Docker images (Pull request #1644) by @bouweandela
- Update documentation on ESMValTool module at DKRZ (Pull request #1647) by @remi-kazoni
- Expanded information on deprecations in changelog (Pull request #1658) by @schlunma

#### 10.12.8 Improvements

- Removed trailing whitespace in custom CMOR tables (Pull request #1564) by @schlunma
- Try searching multiple ESGF index nodes (Pull request #1561) by @bouweandela
- Add CMIP6 *amoc* derivation case and add a test (Pull request #1577) by @valeriupredo
- Added EMAC CMORizer (Pull request #1554) by @schlunma

- Improve performance of *volume\_statistics* (Pull request #1545) by @sloosvel

### 10.12.9 Fixes for datasets

- Fixes of ocean variables in multiple CMIP6 datasets (Pull request #1566) by @tomaslovato
- Ensure lat/lon bounds in FGOALS-I3 atmos variables are contiguous (Pull request #1571) by @sloosvel
- Added *AllVars* fix for CMIP6's ICON-ESM-LR (Pull request #1582) by @schlunma

### 10.12.10 Installation

- Removed *package/meta.yml* (Pull request #1540) by @schlunma
- Pinned iris>=3.2.1 (Pull request #1552) by @schlunma
- Use setuptools-scm to automatically generate the version number (Pull request #1578) by @bouweandela
- Pin cf-units to lower than 3.1.0 to temporarily avoid changes within new version related to calendars (Pull request #1659) by @valeriupredoi

### 10.12.11 Preprocessor

- Allowed special case for unit conversion of precipitation (*kg m<sup>-2</sup> s<sup>-1</sup> <-> mm day<sup>-1</sup>*) (Pull request #1574) by @schlunma
- Add general *extract\_coordinate\_points* preprocessor (Pull request #1581) by @sloosvel
- Add preprocessor *accumulate\_coordinate* (Pull request #1281) by @jvegreg
- Add *axis\_statistics* and improve *depth\_integration* (Pull request #1589) by @sloosvel

### 10.12.12 Release

- Increase version number for ESMValCore v2.6.0rc1 (Pull request #1632) by @sloosvel
- Update changelog and version for 2.6rc3 (Pull request #1646) by @sloosvel
- Add changelog for rc4 (Pull request #1662) by @sloosvel

### 10.12.13 Automatic testing

- Refresh CircleCI cache weekly (Pull request #1597) by @bouweandela
- Use correct cache restore key on CircleCI (Pull request #1598) by @bouweandela
- Install git and ssh before checking out code on CircleCI (Pull request #1601) by @bouweandela
- Fetch all history in Github Action tests (Pull request #1622) by @sloosvel
- Test Github Actions dashboard badge from meercode.io (Pull request #1640) by @valeriupredoi
- Improve esmvalcore.esgf unit test (Pull request #1650) by @bouweandela

### 10.12.14 Variable Derivation

- Added derivation of *hfns* (Pull request #1594) by @schlunma

## 10.13 v2.5.0

### 10.13.1 Highlights

- The new preprocessor `extract_location()` can extract arbitrary locations on the Earth using the `geopy` package that connects to OpenStreetMap. For details, see [Extract location](#).
- Time ranges can now be extracted using the ISO 8601 format. In addition, wildcards are allowed, which makes the time selection much more flexible. For details, see [Recipe section: Datasets](#).
- The new preprocessor `ensemble_statistics()` can calculate arbitrary statistics over all ensemble members of a simulation. In addition, the preprocessor `multi_model_statistics()` now accepts the keyword `groupy`, which allows the calculation of multi-model statistics over arbitrary multi-model ensembles. For details, see [Ensemble statistics](#) and [Multi-model statistics](#).

This release includes

### 10.13.2 Backwards incompatible changes

- Update Cordex section in `config-developer.yml` (Pull request #1303) by @francesco-cmcc. This changes the naming convention of ESMValCore's output files from CORDEX dataset. This only affects recipes that use CORDEX data. Most likely, no changes in diagnostics are necessary; however, if code relies on the specific naming convention of files, it might need to be adapted.
- Dropped Python 3.7 (Pull request #1530) by @schlunma. ESMValCore v2.5.0 dropped support for Python 3.7. From now on Python  $\geq 3.8$  is required to install ESMValCore. The main reason for this is that conda-forge dropped support for Python 3.7 for OSX and arm64 (more details are given [here](#)).

### 10.13.3 Bug fixes

- Fix `extract_shape` when fx vars are present (Pull request #1403) by @sloosvel
- Added support of `extra_facets` to fx variables added by the preprocessor (Pull request #1399) by @schlunma
- Augmented input for derived variables with `extra_facets` (Pull request #1412) by @schlunma
- Correctly use masked arrays after `unstructured_nearest` regridding (Pull request #1414) by @schlunma
- Fixing the broken derivation script for XCH4 (and XCO2) (Pull request #1428) by @hb326
- Ignore `.pymon-journal` file in test discovery (Pull request #1436) by @valeriupredo
- Fixed bug that caused automatic download to fail in rare cases (Pull request #1442) by @schlunma
- Add new `JULIA_LOAD_PATH` to diagnostic task test (Pull request #1444) by @valeriupredo
- Fix provenance file permissions (Pull request #1468) by @bouweandela
- Fixed usage of `statistics=std_dev` option in multi-model statistics preprocessors (Pull request #1478) by @schlunma
- Removed scalar coordinates `p0` and `ptop` prior to merge in `multi_model_statistics` (Pull request #1471) by @axel-lauer
- Added `dataset` and `alias` attributes to `multi_model_statistics` output (Pull request #1483) by @schlunma
- Fixed issues with multi-model-statistics timeranges (Pull request #1486) by @schlunma
- Fixed output messages for CMOR logging (Pull request #1494) by @schlunma
- Fixed `clip_timerange` if only a single time point is extracted (Pull request #1497) by @schlunma
- Fixed chunking in `multi_model_statistics` (Pull request #1500) by @schlunma

- Fixed renaming of auxiliary coordinates in *multi\_model\_statistics* if coordinates are equal (Pull request #1502) by @schlunma
- Fixed timerange selection for automatic downloads (Pull request #1517) by @schlunma
- Fixed chunking in *multi\_model\_statistics* (Pull request #1524) by @schlunma

### 10.13.4 Deprecations

- Renamed vertical regridding schemes (Pull request #1429) by @schlunma. Old regridding schemes are supported until v2.7.0. For details, see *Vertical interpolation schemes*.

### 10.13.5 Documentation

- Remove duplicate entries in changelog (Pull request #1391) by @zklaus
- Documentation on how to use HPC central installations (Pull request #1409) by @valeriupredo
- Correct brackets in preprocessor documentation for list of seasons (Pull request #1420) by @bouweandela
- Add Python=3.10 to package info, update Circle CI auto install and documentation for Python=3.10 (Pull request #1432) by @valeriupredo
- Reverted unintentional change in *.zenodo.json* (Pull request #1452) by @schlunma
- Synchronized config-user.yml with version from ESMValTool (Pull request #1453) by @schlunma
- Solved issues in configuration files (Pull request #1457) by @schlunma
- Add direct link to download conda lock file in the install documentation (Pull request #1462) by @valeriupredo
- CITATION.cff fix and automatic validation of citation metadata (Pull request #1467) by @valeriupredo
- Updated documentation on how to deprecate features (Pull request #1426) by @schlunma
- Added reference hook to conda lock in documentation install section (Pull request #1473) by @valeriupredo
- Increased ESMValCore version to 2.5.0rc1 (Pull request #1477) by @schlunma
- Added changelog for v2.5.0 release (Pull request #1476) by @schlunma
- Increased ESMValCore version to 2.5.0rc2 (Pull request #1487) by @schlunma
- Added some authors to citation and zenodo files (Pull request #1488) by @SarahAlidoost
- Restored *scipy* intersphinx mapping (Pull request #1491) by @schlunma
- Increased ESMValCore version to 2.5.0rc3 (Pull request #1504) by @schlunma
- Fix download instructions for the MSWEP dataset (Pull request #1506) by @remi-kazeroni
- Documentation updated for the new cmorizer framework (Pull request #1417) by @remi-kazeroni
- Added tests for duplicates in changelog and removed duplicates (Pull request #1508) by @schlunma
- Increased ESMValCore version to 2.5.0rc4 (Pull request #1519) by @schlunma
- Add Github Actions Test badge in README (Pull request #1526) by @valeriupredo
- Increased ESMValCore version to 2.5.0rc5 (Pull request #1529) by @schlunma
- Increased ESMValCore version to 2.5.0rc6 (Pull request #1532) by @schlunma

### 10.13.6 Fixes for datasets

- Added fix for AIRS v2.1 (obs4mips) (Pull request #1472) by @axel-lauer

### 10.13.7 Preprocessor

- Added bias preprocessor (Pull request #1406) by @schlunma
- Improve error messages when a preprocessor is failing (Pull request #1408) by @schlunma
- Added option to explicitly not use fx variables in preprocessors (Pull request #1416) by @schlunma
- Add *extract\_location* preprocessor to extract town, city, mountains etc - anything specifiable by a location (Pull request #1251) by @jvegreg
- Add ensemble statistics preprocessor and 'groupby' option for multimodel (Pull request #673) by @sloosvel
- Generic regridding preprocessor (Pull request #1448) by @zklaus

### 10.13.8 Automatic testing

- Add *pandas* as dependency :panda\_face: (Pull request #1402) by @valeriupredoi
- Fixed tests for python 3.7 (Pull request #1410) by @schlunma
- Remove accessing *.xml()* cube method from test (Pull request #1419) by @valeriupredoi
- Remove flag to use pip 2020 solver from Github Action pip install command on OSX (Pull request #1357) by @valeriupredoi
- Add Python=3.10 to Github Actions and switch to Python=3.10 for the Github Action that builds the PyPi package (Pull request #1430) by @valeriupredoi
- Pin *flake8*<4 to keep getting relevant error traces when tests fail with FLAKE8 issues (Pull request #1434) by @valeriupredoi
- Implementing conda lock (Pull request #1164) by @valeriupredoi
- Relocate *pytest-monitor* outputted database *.pymon* so *.pymon-journal* file should not be looked for by *pytest* (Pull request #1441) by @valeriupredoi
- Switch to Mambaforge in Github Actions tests (Pull request #1438) by @valeriupredoi
- Turn off conda lock file creation on any push on *main* branch from Github Action test (Pull request #1489) by @valeriupredoi
- Add DRS path test for IPSLCM files (Pull request #1490) by @senesis
- Add a test module that runs tests of *iris* I/O every time we notice serious bugs there (Pull request #1510) by @valeriupredoi
- [Github Actions] Trigger Github Actions tests (*run-tests.yml* workflow) from a comment in a PR (Pull request #1520) by @valeriupredoi
- Update Linux condalock file (various pull requests) github-actions[bot]

### 10.13.9 Installation

- Move *nested-lookup* dependency to *environment.yml* to be installed from conda-forge instead of PyPi (Pull request #1481) by @valeriupredoi
- Pinned *iris* (Pull request #1511) by @schlunma
- Updated dependencies (Pull request #1521) by @schlunma

- Pinned iris<3.2.0 (Pull request #1525) by @schlunma

### 10.13.10 Improvements

- Allow to load all files, first X years or last X years in an experiment (Pull request #1133) by @sloosvel
- Filter tasks earlier (Pull request #1264) by @jvegreg
- Added earlier validation for command line arguments (Pull request #1435) by @schlunma
- Remove *profile\_diagnostic* from diagnostic settings and increase test coverage of *\_task.py* (Pull request #1404) by @valeriupredo
- Add *output2* to the *product* extra facet of CMIP5 data (Pull request #1514) by @remi-kazoni
- Speed up ESGF search (Pull request #1512) by @bouweandela

## 10.14 v2.4.0

### 10.14.1 Highlights

- ESMValCore now has the ability to automatically download missing data from ESGF. For details, see *Data Retrieval*.
- ESMValCore now also can resume an earlier run. This is useful to reuse expensive preprocessor results. For details, see *Running*.

This release includes

### 10.14.2 Bug fixes

- Crop on the ID-selected region(s) and not on the whole shapefile (Pull request #1151) by @stefsmets
- Add 'comment' to list of removed attributes (Pull request #1244) by @Peter9192
- Speed up multimodel statistics and fix bug in peak computation (Pull request #1301) by @bouweandela
- No longer make plots of provenance (Pull request #1307) by @bouweandela
- No longer embed provenance in output files (Pull request #1306) by @bouweandela
- Removed automatic addition of areacello to obs4mips datasets (Pull request #1316) by @schlunma
- Pin docutils <0.17 to fix bullet lists on readthedocs (Pull request #1320) by @zklaus
- Fix obs4MIPs capitalization (Pull request #1328) by @bouweandela
- Fix Python 3.7 tests (Pull request #1330) by @bouweandela
- Handle fx variables in *extract\_levels* and some time operations (Pull request #1269) by @sloosvel
- Refactored mask regridding for irregular grids (fixes #772) (Pull request #865) by @zklaus
- Fix *da.broadcast\_to* call when the fx cube has different shape than target data cube (Pull request #1350) by @valeriupredo
- Add tests for *\_aggregate\_time\_fx* (Pull request #1354) by @sloosvel
- Fix extra facets (Pull request #1360) by @bouweandela
- Pin pip!=21.3 to avoid pypa/pip#10573 with editable installs (Pull request #1359) by @zklaus
- Add a custom *date2num* function to deal with changes in cftime (Pull request #1373) by @zklaus
- Removed custom version of *AtmosphereSigmaFactory* (Pull request #1382) by @schlunma

### 10.14.3 Deprecations

- Remove `write_netcdf` and `write_plots` from `config-user.yml` (Pull request #1300) by @bouweandela

### 10.14.4 Documentation

- Add link to plot directory in `index.html` (Pull request #1256) by @stefsmets
- Work around issue with yapf not following PEP8 (Pull request #1277) by @bouweandela
- Update the core development team (Pull request #1278) by @bouweandela
- Update the documentation of the provenance interface (Pull request #1305) by @bouweandela
- Update version number to first release candidate 2.4.0rc1 (Pull request #1363) by @zklaus
- Update to new ESMValTool logo (Pull request #1374) by @zklaus
- Update version number for third release candidate 2.4.0rc3 (Pull request #1384) by @zklaus
- Update changelog for 2.4.0rc3 (Pull request #1385) by @zklaus
- Update version number to final 2.4.0 release (Pull request #1389) by @zklaus
- Update changelog for 2.4.0 (Pull request #1366) by @zklaus

### 10.14.5 Fixes for datasets

- Add fix for differing latitude coordinate between historical and `ssp585` in `MPI-ESM1-2-HR r2i1p1f1` (Pull request #1292) by @bouweandela
- Add fixes for time and latitude coordinate of `EC-Earth3 r3i1p1f1` (Pull request #1290) by @bouweandela
- Apply latitude fix to all `CCSM4` variables (Pull request #1295) by @bouweandela
- Fix lat and lon bounds for `FGOALS-g3 mrsos` (Pull request #1289) by @thomascrocker
- Add grid fix for `tos` in `fgoals-f3-l` (Pull request #1326) by @sloosvel
- Add fix for `CIESM pr` (Pull request #1344) by @bouweandela
- Fix `DRS` for `IPSLCM` : split attribute 'freq' into : 'out' and 'freq' (Pull request #1304) by @senesis

### 10.14.6 CMOR standard

- Remove history attribute from `coords` (Pull request #1276) by @jvegreg
- Increased flexibility of CMOR checks for datasets with generic `alevel` coordinates (Pull request #1032) by @schlunma
- Automatically fix small deviations in vertical levels (Pull request #1177) by @bouweandela
- Adding standard names to the custom tables of the `rlns` and `rsns` variables (Pull request #1386) by @remi-kazeroni

### 10.14.7 Preprocessor

- Implemented fully lazy `climate_statistics` (Pull request #1194) by @schlunma
- Run the multimodel statistics preprocessor last (Pull request #1299) by @bouweandela

### 10.14.8 Automatic testing

- Improving test coverage for `_task.py` (Pull request #514) by @valeriupredoi
- Upload coverage to codecov (Pull request #1190) by @bouweandela
- Improve codecov status checks (Pull request #1195) by @bouweandela
- Fix curl install in CircleCI (Pull request #1228) by @jvegreg
- Drop support for Python 3.6 (Pull request #1200) by @valeriupredoi
- Allow more recent version of *scipy* (Pull request #1182) by @schlunma
- Speed up conda build *conda\_build* Circle test by using *mamba* solver via *boa* (and use it for Github Actions test too) (Pull request #1243) by @valeriupredoi
- Fix numpy deprecation warnings (Pull request #1274) by @bouweandela
- Unpin upper bound for iris (previously was at <3.0.4) (Pull request #1275) by @valeriupredoi
- Modernize *conda\_install* test on Circle CI by installing from conda-forge with Python 3.9 and change install instructions in documentation (Pull request #1280) by @valeriupredoi
- Run a nightly Github Actions workflow to monitor tests memory per test (configurable for other metrics too) (Pull request #1284) by @valeriupredoi
- Speed up tests of tasks (Pull request #1302) by @bouweandela
- Fix upper case to lower case variables and functions for flake compliance in *tests/unit/preprocessor/\_regrid/test\_extract\_levels.py* (Pull request #1347) by @valeriupredoi
- Cleaned up a bit Github Actions workflows (Pull request #1345) by @valeriupredoi
- Update circleci jobs: renaming tests to more descriptive names and removing conda build test (Pull request #1351) by @zklaus
- Pin iris to latest  $\geq 3.1.0$  (Pull request #1341) by @valeriupredoi

### 10.14.9 Installation

- Pin esmpy to anything but 8.1.0 since that particular one changes the CPU affinity (Pull request #1310) by @valeriupredoi

### 10.14.10 Improvements

- Add a more friendly and useful message when using default config file (Pull request #1233) by @valeriupredoi
- Replace `os.walk` by `glob.glob` in data finder (only look for data in the specified locations) (Pull request #1261) by @bouweandela
- Machine-specific directories for auxiliary data in the *config-user.yml* file (Pull request #1268) by @remi-kazoni
- Add an option to download missing data from ESGF (Pull request #1217) by @bouweandela
- Speed up provenance recording (Pull request #1327) by @bouweandela
- Improve results web page (Pull request #1332) by @bouweandela
- Move institutes from *config-developer.yml* to default extra facets config and add wildcard support for extra facets (Pull request #1259) by @bouweandela
- Add support for reusing preprocessor output from previous runs (Pull request #1321) by @bouweandela
- Log fewer messages to screen and hide stack trace for known recipe errors (Pull request #1296) by @bouweandela
- Log ESMValCore and ESMValTool versions when running (Pull request #1263) by @jvegreg

- Add “grid” as a tag to the output file template for CMIP6 (Pull request #1356) by @zklaus
- Implemented ICON project to read native ICON model output (Pull request #1079) by @bsolino

## 10.15 v2.3.1

This release includes

### 10.15.1 Bug fixes

- Update config-user.yml template with correct drs entries for CEDA-JASMIN (Pull request #1184) by @valeriupredo
- Enhancing MIROC5 fix for hfls and evspsbl (Pull request #1192) by @katjaweigel
- Fix alignment of daily data with inconsistent calendars in multimodel statistics (Pull request #1212) by @Peter9192
- Pin cf-units, remove github actions test for Python 3.6 and fix test\_access1\_0 and test\_access1\_3 to use cf-units for comparisons (Pull request #1197) by @valeriupredo
- Fixed search for fx files when no mip is given (Pull request #1216) by @schlunma
- Make sure climate statistics always returns original dtype (Pull request #1237) by @zklaus
- Bugfix for regional regridding when non-integer range is passed (Pull request #1231) by @stefsmets
- Make sure area\_statistics preprocessor always returns original dtype (Pull request #1239) by @zklaus
- Add “.” (dot) as allowed separation character for the time range group (Pull request #1248) by @zklaus

### 10.15.2 Documentation

- Add a link to the instructions to use pre-installed versions on HPC clusters (Pull request #1186) by @remikazeroni
- Bugfix release: set version to 2.3.1 (Pull request #1253) by @zklaus

### 10.15.3 Fixes for datasets

- Set circular attribute in MCM-UA-1-0 fix (Pull request #1178) by @sloosvel
- Fixed time coordinate of MIROC-ESM (Pull request #1188) by @schlunma

### 10.15.4 Preprocessor

- Filter warnings about collapsing multi-model dimension in multimodel statistics preprocessor function (Pull request #1215) by @bouweandela
- Remove fx variables before computing multimodel statistics (Pull request #1220) by @sloosvel

### 10.15.5 Installation

- Pin lower bound for iris to 3.0.2 (Pull request #1206) by @valeriupredo
- Pin *iris*<3.0.4 to ensure we still (sort of) support Python 3.6 (Pull request #1252) by @valeriupredo

### 10.15.6 Improvements

- Add test to verify behaviour for scalar height coord for tas in multi-model (Pull request #1209) by @Peter9192
- Sort missing years in “No input data available for years” message (Pull request #1225) by @ledm

## 10.16 v2.3.0

This release includes

### 10.16.1 Bug fixes

- Extend preprocessor multi\_model\_statistics to handle data with “altitude” coordinate (Pull request #1010) by @axel-lauer
- Remove scripts included with CMOR tables (Pull request #1011) by @bouweandela
- Avoid side effects in extract\_season (Pull request #1019) by @jvegreg
- Use nearest scheme to avoid interpolation errors with masked data in regression test (Pull request #1021) by @stefsmeets
- Move \_get\_time\_bounds from preprocessor.\_time to cmor.check to avoid circular import with cmor module (Pull request #1037) by @valeriupredo
- Fix test that makes conda build fail (Pull request #1046) by @valeriupredo
- Fix ‘positive’ attribute for rsn/rlns variables (Pull request #1051) by @lukasbrunner
- Added preprocessor mask\_multimodel (Pull request #767) by @schlunma
- Fix bug when fixing bounds after fixing longitude values (Pull request #1057) by @sloosvel
- Run conda build parallel AND sequential tests (Pull request #1065) by @valeriupredo
- Add key to id\_prop (Pull request #1071) by @lukasbrunner
- Fix bounds after reversing coordinate values (Pull request #1061) by @sloosvel
- Fixed –skip-nonexistent option (Pull request #1093) by @schlunma
- Do not consider CMIP5 variable sit to be the same as sithick from CMIP6 (Pull request #1033) by @bouweandela
- Improve finding date range in filenames (enforces separators) (Pull request #1145) by @senesis
- Review fx handling (Pull request #1147) by @sloosvel
- Fix lru cache decorator with explicit call to method (Pull request #1172) by @valeriupredo
- Update \_volume.py (Pull request #1174) by @ledm

### 10.16.2 Deprecations

### 10.16.3 Documentation

- Final changelog for 2.3.0 (Pull request #1163) by @zklaus
- Set version to 2.3.0 (Pull request #1162) by @zklaus
- Fix documentation build (Pull request #1006) by @bouweandela
- Add labels required for linking from ESMValTool docs (Pull request #1038) by @bouweandela
- Update contribution guidelines (Pull request #1047) by @bouweandela
- Fix basestring references in documentation (Pull request #1106) by @jvegreg

- Updated references master to main (Pull request #1132) by @axel-lauer
- Add instructions how to use the central installation at DKRZ-Mistral (Pull request #1155) by @remi-kazoni

### 10.16.4 Fixes for datasets

- Added fixes for various CMIP5 datasets, variable cl (3-dim cloud fraction) (Pull request #1017) by @axel-lauer
- Added fixes for hybrid level coordinates of CESM2 models (Pull request #882) by @schlunma
- Extending LWP fix for CMIP6 models (Pull request #1049) by @axel-lauer
- Add fixes for the net & up radiation variables from ERA5 (Pull request #1052) by @lukasbrunner
- Add derived variable rsus (Pull request #1053) by @lukasbrunner
- Supported *mip*-level fixes (Pull request #1095) by @schlunma
- Fix erroneous use of *grid\_latitude* and *grid\_longitude* and cleaned ocean grid fixes (Pull request #1092) by @schlunma
- Fix for pr of miroc5 (Pull request #1110) by @remi-kazoni
- Ocean depth fix for cnrm\_esm2\_1, gfdl\_esm4, ipsl\_cm6a\_lr datasets + mcm\_ua\_1\_0 (Pull request #1098) by @tomaslovato
- Fix for uas variable of the MCM\_UA\_1\_0 dataset (Pull request #1102) by @remi-kazoni
- Fixes for sos and siconc of BCC models (Pull request #1090) by @remi-kazoni
- Run fgco2 fix for all CESM2 models (Pull request #1108) by @LisaBock
- Fixes for the siconc variable of CMIP6 models (Pull request #1105) by @remi-kazoni
- Fix wrong sign for land surface flux (Pull request #1113) by @LisaBock
- Fix for pr of EC\_EARTH (Pull request #1116) by @remi-kazoni

### 10.16.5 CMOR standard

- Format cmor related files (Pull request #976) by @jvegreg
- Check presence of time bounds and guess them if needed (Pull request #849) by @sloosvel
- Add custom variable “tasaga” (Pull request #1118) by @LisaBock
- Find files for CMIP6 DCPD startdates (Pull request #771) by @sloosvel

### 10.16.6 Preprocessor

- Update tests for multimodel statistics preprocessor (Pull request #1023) by @stefsmeets
- Raise in extract\_season and extract\_month if result is None (Pull request #1041) by @jvegreg
- Allow selection of shapes in extract\_shape (Pull request #764) by @jvegreg
- Add option for regional regridding to regrid preprocessor (Pull request #1034) by @stefsmeets
- Load fx variables as cube cell measures / ancillary variables (Pull request #999) by @sloosvel
- Check horizontal grid before regridding (Pull request #507) by @BenMGeo
- Clip irregular grids (Pull request #245) by @bouweandela
- Use native iris functions in multi-model statistics (Pull request #1150) by @Peter9192

### 10.16.7 Notebook API (experimental)

### 10.16.8 Automatic testing

- Report coverage for tests that run on any pull request (Pull request #994) by @bouweandela
- Install ESMValTool sample data from PyPI (Pull request #998) by @jvegreg
- Fix tests for multi-processing with spawn method (i.e. macOSX with Python>3.8) (Pull request #1003) by @bvreede
- Switch to running the Github Action test workflow every 3 hours in single thread mode to observe if Segmentation Faults occur (Pull request #1022) by @valeriupredo
- Revert to original Github Actions test workflow removing the 3-hourly test run with -n 1 (Pull request #1025) by @valeriupredo
- Avoid stale cache for multimodel statistics regression tests (Pull request #1030) by @bouweandela
- Add newer Python versions in OSX to Github Actions (Pull request #1035) by @bvreede
- Add tests for type annotations with mypy (Pull request #1042) by @stefsmets
- Run problematic cmor tests sequentially to avoid segmentation faults on CircleCI (Pull request #1064) by @valeriupredo
- Test installation of esmvalcore from conda-forge (Pull request #1075) by @valeriupredo
- Added additional test cases for integration tests of data\_finder.py (Pull request #1087) by @schlunma
- Pin cf-units and fix tests (cf-units>=2.1.5) (Pull request #1140) by @valeriupredo
- Fix failing CircleCI tests (Pull request #1167) by @bouweandela
- Fix test failing due to fx files chosen differently on different OS's (Pull request #1169) by @valeriupredo
- Compare datetimes instead of strings in \_fixes/cmip5/test\_access1\_X.py (Pull request #1173) by @valeriupredo
- Pin Python to 3.9 in environment.yml on CircleCI and skip mypy tests in conda build (Pull request #1176) by @bouweandela

### 10.16.9 Installation

- Update yamale to version 3 (Pull request #1059) by @zklaus

### 10.16.10 Improvements

- Refactor diagnostics / tags management (Pull request #939) by @stefsmets
- Support multiple paths in input\_dir (Pull request #1000) by @jvegreg
- Generate HTML report with recipe output (Pull request #991) by @stefsmets
- Add timeout to requests.get in \_citation.py (Pull request #1091) by @SarahAlidoost
- Add SYNDA drs for CMIP5 and CMIP6 (closes #582) (Pull request #583) by @zklaus
- Add basic support for variable mappings (Pull request #1124) by @zklaus
- Handle IPSL-CM6 (Pull request #1153) by @senesis

## 10.17 v2.2.0

### 10.17.1 Highlights

ESMValCore is now using the recently released [Iris 3](#). We acknowledge that this change may impact your work, as Iris 3 introduces several changes that are not backward-compatible, but we think that moving forward is the best decision for the tool in the long term.

This release is also the first one including support for downloading CMIP6 data using Synda and we have also started supporting Python 3.9. Give it a try!

This release includes

### 10.17.2 Bug fixes

- Fix path settings for DKRZ/Mistral (Pull request #852) by @bouweandela
- Change logic for calling the diagnostic script to avoid problems with scripts where the executable bit is accidentally set (Pull request #877) by @bouweandela
- Fix overwriting in generic level check (Pull request #886) by @sloosvel
- Add double quotes to script args in rerun screen message when using vprof profiling (Pull request #897) by @valeriupredo
- Simplify time handling in multi-model statistics preprocessor (Pull request #685) by @Peter9192
- Fix links to Iris documentation (Pull request #966) by @jvegreg
- Bugfix: Fix units for MSWEP data (Pull request #986) by @stefsmets

### 10.17.3 Deprecations

- Deprecate defining write\_plots and write\_netcdf in config-user file (Pull request #808) by @bouweandela

### 10.17.4 Documentation

- Fix numbering of steps in release instructions (Pull request #838) by @bouweandela
- Add labels to changelogs of individual versions for easy reference (Pull request #899) by @zklaus
- Make CircleCI badge specific to main branch (Pull request #902) by @bouweandela
- Fix docker build badge url (Pull request #906) by @stefsmets
- Update github PR template (Pull request #909) by @stefsmets
- Refer to ESMValTool GitHub discussions page in the error message (Pull request #900) by @bouweandela
- Support automatically closing issues (Pull request #922) by @bouweandela
- Fix checkboxes in PR template (Pull request #931) by @stefsmets
- Change in config-user defaults and documentation with new location for esmeval OBS data on JASMIN (Pull request #958) by @valeriupredo
- Update Core Team info (Pull request #942) by @axel-lauer
- Update iris documentation URL for sphinx (Pull request #964) by @bouweandela
- Set version to 2.2.0 (Pull request #977) by @jvegreg
- Add first draft of v2.2.0 changelog (Pull request #983) by @jvegreg
- Add checkbox in PR template to assign labels (Pull request #985) by @jvegreg

- Update install.rst (Pull request #848) by @bascrezee
- Change the order of the publication steps (Pull request #984) by @jvegreg
- Add instructions how to use esmvaltool from HPC central installations (Pull request #841) by @valeriupredoi

### 10.17.5 Fixes for datasets

- Fixing unit for derived variable rsnstcsnorm to prevent overcorrection2 (Pull request #846) by @katjaweigel
- C mip6 fix awi cm 1 1 mr (Pull request #822) by @mwjury
- C mip6 fix ec earth3 veg (Pull request #836) by @mwjury
- Changed latitude longitude fix from Tas to AllVars. (Pull request #916) by @katjaweigel
- Fix for precipitation (pr) to use ERA5-Land cmorizer (Pull request #879) by @katjaweigel
- C mip6 fix ec earth3 (Pull request #837) by @mwjury
- C mip6\_fix\_fgoals\_f3\_l\_Amon\_time\_bnds (Pull request #831) by @mwjury
- Fix for FGOALS-f3-L sftlf (Pull request #667) by @mwjury
- Improve ACCESS-CM2 and ACCESS-ESM1-5 fixes and add CIESM and CESM2-WACCM-FV2 fixes for cl, clw and cli (Pull request #635) by @axel-lauer
- Add fixes for cl, cli, clw and tas for several CMIP6 models (Pull request #955) by @schlunma
- Dataset fixes for MSWEP (Pull request #969) by @stefsmeets
- Dataset fixes for: ACCESS-ESM1-5, CanESM5, CanESM5 for carbon cycle (Pull request #947) by @bettina-gier
- Fixes for KIOST-ESM (CMIP6) (Pull request #904) by @remi-kazoni
- Fixes for AWI-ESM-1-1-LR (CMIP6, piControl) (Pull request #911) by @remi-kazoni

### 10.17.6 CMOR standard

- CMOR check generic level coordinates in CMIP6 (Pull request #598) by @sloosvel
- Update CMIP6 tables to 6.9.33 (Pull request #919) by @jvegreg
- Adding custom variables for tas uncertainty (Pull request #924) by @LisaBock
- Remove monotonicity coordinate check for unstructured grids (Pull request #965) by @jvegreg

### 10.17.7 Preprocessor

- Added clip\_start\_end\_year preprocessor (Pull request #796) by @schlunma
- Add support for downloading CMIP6 data with Synda (Pull request #699) by @bouweandela
- Add multimodel tests using real data (Pull request #856) by @stefsmeets
- Add plev/altitude conversion to extract\_levels (Pull request #892) by @axel-lauer
- Add possibility of custom season extraction. (Pull request #247) by @mwjury
- Adding the ability to derive xch4 (Pull request #783) by @hb326
- Add preprocessor function to resample time and compute x-hourly statistics (Pull request #696) by @jvegreg
- Fix duplication in preprocessors DEFAULT\_ORDER introduced in #696 (Pull request #973) by @jvegreg

- Use consistent precision in multi-model statistics calculation and update reference data for tests (Pull request #941) by @Peter9192
- Refactor multi-model statistics code to facilitate ensemble stats and lazy evaluation (Pull request #949) by @Peter9192
- Add option to exclude input cubes in output of multimodel statistics to solve an issue introduced by #949 (Pull request #978) by @Peter9192

### 10.17.8 Automatic testing

- Pin cftime $\geq$ 1.3.0 to have newer string formatting in and fix two tests (Pull request #878) by @valeriupredoi
- Switched miniconda conda setup hooks for Github Actions workflows (Pull request #873) by @valeriupredoi
- Add test for latest version resolver (Pull request #874) by @stefsmeeets
- Update codacy coverage reporter to fix coverage (Pull request #905) by @nielsdorst
- Avoid hardcoded year in tests and add improvement to plev test case (Pull request #921) by @bouweandela
- Pin scipy to less than 1.6.0 until Issue #927 gets resolved (Pull request #928) by @valeriupredoi
- Github Actions: change time when conda install test runs (Pull request #930) by @valeriupredoi
- Remove redundant test line from test\_utils.py (Pull request #935) by @valeriupredoi
- Removed netCDF4 package from integration tests of fixes (Pull request #938) by @schlunma
- Use new conda environment for installing ESMValCore in Docker containers (Pull request #951) by @bouweandela

### 10.17.9 Notebook API (experimental)

- Implement importable config object in experimental API submodule (Pull request #868) by @stefsmeeets
- Add loading and running recipes to the notebook API (Pull request #907) by @stefsmeeets
- Add displaying and loading of recipe output to the notebook API (Pull request #957) by @stefsmeeets
- Add functionality to run single diagnostic task to notebook API (Pull request #962) by @stefsmeeets

### 10.17.10 Improvements

- Create CODEOWNERS file (Pull request #809) by @jvegreg
- Remove code needed for Python <3.6 (Pull request #844) by @bouweandela
- Add requests as a dependency (Pull request #850) by @bouweandela
- Pin Python to less than 3.9 (Pull request #870) by @valeriupredoi
- Remove numba dependency (Pull request #880) by @schlunma
- Add Listing and finding recipes to the experimental notebook API (Pull request #901) by @stefsmeeets
- Skip variables that don't have dataset or additional\_dataset keys (Pull request #860) by @valeriupredoi
- Refactor logging configuration (Pull request #933) by @stefsmeeets
- Xco2 derivation (Pull request #913) by @bettina-gier
- Working environment for Python 3.9 (pin to  $\neq$ 3.9.0) (Pull request #885) by @valeriupredoi
- Print source file when using config get\_config\_user command (Pull request #960) by @valeriupredoi
- Switch to Iris 3 (Pull request #819) by @stefsmeeets

- Refactor tasks (Pull request #959) by @stefsmeeets
- Restore task summary in debug log after #959 (Pull request #981) by @bouweandela
- Pin pre-commit hooks (Pull request #974) by @stefsmeeets
- Improve error messages when data is missing (Pull request #917) by @jvegreg
- Set remove\_preproc\_dir to false in default config-user (Pull request #979) by @valeriupredo
- Move fiona to be installed from conda forge (Pull request #987) by @valeriupredo
- Re-added fiona in setup.py (Pull request #990) by @valeriupredo

## 10.18 v2.1.0

This release includes

### 10.18.1 Bug fixes

- Set unit=1 if anomalies are standardized (Pull request #727) by @bascrezee
- Fix crash for FGOALS-g2 variables without longitude coordinate (Pull request #729) by @bouweandela
- Improve variable alias management (Pull request #595) by @jvegreg
- Fix area\_statistics fx files loading (Pull request #798) by @jvegreg
- Fix units after derivation (Pull request #754) by @schlunma

### 10.18.2 Documentation

- Update v2.0.0 release notes with final additions (Pull request #722) by @bouweandela
- Update package description in setup.py (Pull request #725) by @mattiarighi
- Add installation instructions for pip installation (Pull request #735) by @bouweandela
- Improve config-user documentation (Pull request #740) by @bouweandela
- Update the zenodo file with contributors (Pull request #807) by @valeriupredo
- Improve command line run documentation (Pull request #721) by @jvegreg
- Update the zenodo file with contributors (continued) (Pull request #810) by @valeriupredo

### 10.18.3 Improvements

- Reduce size of docker image (Pull request #723) by @jvegreg
- Add 'test' extra to installation, used by docker development tag (Pull request #733) by @bouweandela
- Correct dockerhub link (Pull request #736) by @bouweandela
- Create action-install-from-pypi.yml (Pull request #734) by @valeriupredo
- Add pre-commit for linting/formatting (Pull request #766) by @stefsmeeets
- Run tests in parallel and when building conda package (Pull request #745) by @bouweandela
- Readable exclude pattern for pre-commit (Pull request #770) by @stefsmeeets
- Github Actions Tests (Pull request #732) by @valeriupredo
- Remove isort setup to fix formatting conflict with yapf (Pull request #778) by @stefsmeeets

- Fix yapf-isort import formatting conflict (Fixes #777) (Pull request #784) by @stefsmeets
- Sorted output for *esmvaltool recipes list* (Pull request #790) by @stefsmeets
- Replace vmprof with vprof (Pull request #780) by @valeriupredo
- Update CMIP6 tables to 6.9.32 (Pull request #706) by @jvegreg
- Default config-user path now set in config-user read function (Pull request #791) by @jvegreg
- Add custom variable lweGrace (Pull request #692) by @bascrezee
- Create Github Actions workflow to build and deploy on Test PyPi and PyPi (Pull request #820) by @valeriupredo
- Build and publish the esmvalcore package to conda via Github Actions workflow (Pull request #825) by @valeriupredo

#### 10.18.4 Fixes for datasets

- Fix cmip6 models (Pull request #629) by @npgillett
- Fix siconca variable in EC-Earth3 and EC-Earth3-Veg models in amip simulation (Pull request #702) by @egalytska

#### 10.18.5 Preprocessor

- Move cmor\_check\_data to early in preprocessing chain (Pull request #743) by @bouweandela
- Add RMS iris analysis operator to statistics preprocessor functions (Pull request #747) by @pcosbsc
- Add surface chlorophyll concentration as a derived variable (Pull request #720) by @sloosvel
- Use dask to reduce memory consumption of extract\_levels for masked data (Pull request #776) by @valeriupredo

### 10.19 v2.0.0

This release includes

#### 10.19.1 Bug fixes

- Fixed derivation of co2s (Pull request #594) by @schlunma
- Padding while cropping needs to stay within sane bounds for shapefiles that span the whole Earth (Pull request #626) by @valeriupredo
- Fix concatenation of a single cube (Pull request #655) by @bouweandela
- Fix mask fx dict handling not to fail if empty list in values (Pull request #661) by @valeriupredo
- Preserve metadata during anomalies computation when using iris cubes difference (Pull request #652) by @valeriupredo
- Avoid crashing when there is directory 'esmvaltool' in the current working directory (Pull request #672) by @valeriupredo
- Solve bug in ACCESS1 dataset fix for calendar. (Pull request #671) by @Peter9192
- Fix the syntax for adding multiple ensemble members from the same dataset (Pull request #678) by @SarahAli-doost
- Fix bug that made preprocessor with fx files fail in rare cases (Pull request #670) by @schlunma
- Add support for string coordinates (Pull request #657) by @jvegreg

- Fixed the shape extraction to account for wraparound shapefile coords (Pull request #319) by @valeriupredoi
- Fixed bug in time weights calculation (Pull request #695) by @schlunma
- Fix diagnostic filter (Pull request #713) by @jvegreg

### 10.19.2 Documentation

- Add pandas as a requirement for building the documentation (Pull request #607) by @bouweandela
- Document default order in which preprocessor functions are applied (Pull request #633) by @bouweandela
- Add pointers about data loading and CF standards to documentation (Pull request #571) by @valeriupredoi
- Config file populated with site-specific data paths examples (Pull request #619) by @valeriupredoi
- Update Codacy badges (Pull request #643) by @bouweandela
- Update copyright info on readthedocs (Pull request #668) by @bouweandela
- Updated references to documentation (now docs.esmvaltool.org) (Pull request #675) by @axel-lauer
- Add all European grants to Zenodo (Pull request #680) by @bouweandela
- Update Sphinx to v3 or later (Pull request #683) by @bouweandela
- Increase version to 2.0.0 and add release notes (Pull request #691) by @bouweandela
- Update setup.py and README.md for use on PyPI (Pull request #693) by @bouweandela
- Suggested Documentation changes (Pull request #690) by @ssmithClimate

### 10.19.3 Improvements

- Reduce the size of conda package (Pull request #606) by @bouweandela
- Add a few unit tests for DiagnosticTask (Pull request #613) by @bouweandela
- Make ncl or R tests not fail if package not installed (Pull request #610) by @valeriupredoi
- Pin flake8<3.8.0 (Pull request #623) by @valeriupredoi
- Log warnings for likely errors in provenance record (Pull request #592) by @bouweandela
- Unpin flake8 (Pull request #646) by @bouweandela
- More flexible native6 default DRS (Pull request #645) by @bouweandela
- Try to use the same python for running diagnostics as for esmvaltool (Pull request #656) by @bouweandela
- Fix test for lower python version and add note on lxml (Pull request #659) by @valeriupredoi
- Added 1m deep average soil moisture variable (Pull request #664) by @bascrezee
- Update docker recipe (Pull request #603) by @jvegreg
- Improve command line interface (Pull request #605) by @jvegreg
- Remove utils directory (Pull request #697) by @bouweandela
- Avoid pytest version that crashes (Pull request #707) by @bouweandela
- Options arg in read\_config\_user\_file now optional (Pull request #716) by @jvegreg
- Produce a readable warning if ancestors are a string instead of a list. (Pull request #711) by @katjaweigel
- Pin Yamale to v2 (Pull request #718) by @bouweandela
- Expanded cmor public API (Pull request #714) by @schlunma

### 10.19.4 Fixes for datasets

- Added various fixes for hybrid height coordinates (Pull request #562) by @schlunma
- Extended fix for cl-like variables of CESM2 models (Pull request #604) by @schlunma
- Added fix to convert “geopotential” to “geopotential height” for ERA5 (Pull request #640) by @egalytska
- Do not fix longitude values if they are too far from valid range (Pull request #636) by @jvegreg

### 10.19.5 Preprocessor

- Implemented concatenation of cubes with derived coordinates (Pull request #546) by @schlunma
- Fix derived variable ctotal calculation depending on project and standard name (Pull request #620) by @valeriupredo
- State of the art FX variables handling without preprocessing (Pull request #557) by @valeriupredo
- Add max, min and std operators to multimodel (Pull request #602) by @jvegreg
- Added preprocessor to extract amplitude of cycles (Pull request #597) by @schlunma
- Overhaul concatenation and allow for correct concatenation of multiple overlapping datasets (Pull request #615) by @valeriupredo
- Change volume stats to handle and output masked array result (Pull request #618) by @valeriupredo
- Area\_weights for cordex in area\_statistics (Pull request #631) by @mwjury
- Accept cubes as input in multimodel (Pull request #637) by @sloosvel
- Make multimodel work correctly with yearly data (Pull request #677) by @valeriupredo
- Optimize time weights in time preprocessor for climate statistics (Pull request #684) by @valeriupredo
- Add percentiles to multi-model stats (Pull request #679) by @Peter9192

## 10.20 v2.0.0b9

This release includes

### 10.20.1 Bug fixes

- Cast dtype float32 to output from zonal and meridional area preprocessors (Pull request #581) by @valeriupredo

### 10.20.2 Improvements

- Unpin on Python<3.8 for conda package (run) (Pull request #570) by @valeriupredo
- Update pytest installation marker (Pull request #572) by @bouweandela
- Remove vmrh2o (Pull request #573) by @mattiarighi
- Restructure documentation (Pull request #575) by @bouweandela
- Fix mask in land variables for CCSM4 (Pull request #579) by @zklaus
- Fix derive scripts wrt required method (Pull request #585) by @zklaus
- Check coordinates do not have repeated standard names (Pull request #558) by @jvegreg
- Added derivation script for co2s (Pull request #587) by @schlunma
- Adapted custom co2s table to match CMIP6 version (Pull request #588) by @schlunma

- Increase version to v2.0.0b9 (Pull request #593) by @bouweandela
- Add a method to save citation information (Pull request #402) by @SarahAlidoost

For older releases, see the release notes on <https://github.com/ESMValGroup/ESMValCore/releases>.

## **Part V**

# **Indices and tables**



- [genindex](#)
- [search](#)



## PYTHON MODULE INDEX

### e

- `esmvalcore.cmor`, 167
- `esmvalcore.cmor.check`, 167
- `esmvalcore.cmor.fix`, 172
- `esmvalcore.cmor.fixes`, 174
- `esmvalcore.cmor.table`, 175
- `esmvalcore.config`, 199
- `esmvalcore.dataset`, 204
- `esmvalcore.esgf.facets`, 214
- `esmvalcore.exceptions`, 214
- `esmvalcore.experimental.recipe`, 290
- `esmvalcore.experimental.recipe_metadata`, 300
- `esmvalcore.experimental.recipe_output`, 294
- `esmvalcore.experimental.utils`, 303
- `esmvalcore.io`, 216
- `esmvalcore.io.esgf`, 217
- `esmvalcore.io.esgf.facets`, 224
- `esmvalcore.io.intake_esgf`, 225
- `esmvalcore.io.local`, 229
- `esmvalcore.io.protocol`, 232
- `esmvalcore.iris_helpers`, 234
- `esmvalcore.local`, 238
- `esmvalcore.preprocessor`, 243
- `esmvalcore.preprocessor.regrid_schemes`, 284
- `esmvalcore.typing`, 288



## A

- `accumulate_coordinate()` (in module `esmval-core.preprocessor`), 245
  - `activities` (`esmvalcore.cmor.table.CMIP6Info` attribute), 181
  - `activities` (`esmvalcore.cmor.table.Obs4MIPsInfo` attribute), 191
  - `add_altitude_from_plev()` (in module `esmval-core.cmor.fixes`), 174
  - `add_leading_dim_to_cube()` (in module `esmval-core.iris_helpers`), 235
  - `add_note()` (`esmvalcore.experimental.recipe_metadata.RenderError` method), 302
  - `add_plev_from_altitude()` (in module `esmval-core.cmor.fixes`), 174
  - `add_supplementary()` (`esmvalcore.dataset.Dataset` method), 205
  - `add_supplementary_variables()` (in module `esmval-core.preprocessor`), 245
  - `align_metadata()` (in module `esmval-core.preprocessor`), 245
  - `alt_names` (`esmvalcore.cmor.table.CMIP3Info` attribute), 177
  - `alt_names` (`esmvalcore.cmor.table.CMIP5Info` attribute), 179
  - `alt_names` (`esmvalcore.cmor.table.CMIP6Info` attribute), 181
  - `alt_names` (`esmvalcore.cmor.table.CustomInfo` attribute), 186
  - `alt_names` (`esmvalcore.cmor.table.InfoBase` attribute), 187
  - `alt_names` (`esmvalcore.cmor.table.NoInfo` attribute), 189
  - `alt_names` (`esmvalcore.cmor.table.Obs4MIPsInfo` attribute), 191
  - `amplitude()` (in module `esmvalcore.preprocessor`), 246
  - `annual_statistics()` (in module `esmval-core.preprocessor`), 246
  - `anomalies()` (in module `esmvalcore.preprocessor`), 247
  - `area_statistics()` (in module `esmval-core.preprocessor`), 247
  - `args` (`esmvalcore.experimental.recipe_metadata.RenderError` attribute), 302
  - `attributes` (`esmvalcore.io.esgf.ESGFFile` property), 220
  - `attributes` (`esmvalcore.io.intake_esgf.IntakeESGFDataset` property), 229
  - `attributes` (`esmvalcore.io.local.LocalFile` property), 232
  - `attributes` (`esmvalcore.io.protocol.DataElement` attribute), 233
  - `attributes` (`esmvalcore.local.LocalFile` property), 241
  - `augment_facets()` (`esmvalcore.dataset.Dataset` method), 205
  - `authors` (`esmvalcore.experimental.recipe_output.DataFile` property), 294
  - `authors` (`esmvalcore.experimental.recipe_output.ImageFile` property), 296
  - `authors` (`esmvalcore.experimental.recipe_output.OutputFile` property), 297
  - `axis` (`esmvalcore.cmor.table.CoordinateInfo` attribute), 184
  - `axis_statistics()` (in module `esmval-core.preprocessor`), 248
- ## B
- `bias()` (in module `esmvalcore.preprocessor`), 248
- ## C
- `caption` (`esmvalcore.experimental.recipe_output.DataFile` property), 294
  - `caption` (`esmvalcore.experimental.recipe_output.ImageFile` property), 296
  - `caption` (`esmvalcore.experimental.recipe_output.OutputFile` property), 297
  - `catalog` (`esmvalcore.io.intake_esgf.IntakeESGFDataset` attribute), 229
  - `catalog` (`esmvalcore.io.intake_esgf.IntakeESGFDDataSource` attribute), 228
  - `CFG` (in module `esmvalcore.config`), 199
  - `check_data()` (`esmvalcore.cmor.check.CMORCheck` method), 169
  - `check_metadata()` (`esmval-core.cmor.check.CMORCheck` method), 169

- CheckLevels (class in *esmvalcore.cmor.check*), 167  
 citation\_file (esmvalcore.experimental.recipe\_output.DataFile property), 294  
 citation\_file (esmvalcore.experimental.recipe\_output.ImageFile property), 296  
 citation\_file (esmvalcore.experimental.recipe\_output.OutputFile property), 297  
 clear() (esmvalcore.cmor.table.TableInfo method), 193  
 clear() (esmvalcore.config.Config method), 199  
 clear() (esmvalcore.config.Session method), 202  
 clear\_table\_cache() (in module *esmvalcore.cmor.table*), 195  
 climate\_statistics() (in module *esmvalcore.preprocessor*), 249  
 clip() (in module *esmvalcore.preprocessor*), 250  
 clip\_timerange() (in module *esmvalcore.preprocessor*), 250  
 CMIP3Info (class in *esmvalcore.cmor.table*), 176  
 CMIP5Info (class in *esmvalcore.cmor.table*), 178  
 CMIP6Info (class in *esmvalcore.cmor.table*), 180  
 cmor\_check() (in module *esmvalcore.cmor.check*), 172  
 cmor\_check\_data() (in module *esmvalcore.cmor.check*), 171  
 cmor\_check\_data() (in module *esmvalcore.preprocessor*), 251  
 cmor\_check\_metadata() (in module *esmvalcore.cmor.check*), 171  
 cmor\_check\_metadata() (in module *esmvalcore.preprocessor*), 251  
 cmor\_log (esmvalcore.config.Session property), 202  
 CMOR\_TABLES (in module *esmvalcore.cmor.table*), 182  
 CMORCheck (class in *esmvalcore.cmor.check*), 168  
 CMORCheckError, 168  
 concatenate() (in module *esmvalcore.preprocessor*), 252  
 Config (class in *esmvalcore.config*), 199  
 context() (esmvalcore.config.Config method), 199  
 context() (esmvalcore.config.Session method), 202  
 Contributor (class in *esmvalcore.experimental.recipe\_metadata*), 300  
 convert\_units() (in module *esmvalcore.preprocessor*), 252  
 CoordinateInfo (class in *esmvalcore.cmor.table*), 184  
 coordinates (esmvalcore.cmor.table.VariableInfo attribute), 194  
 coords (esmvalcore.cmor.table.CMIP3Info attribute), 177  
 coords (esmvalcore.cmor.table.CMIP5Info attribute), 179  
 coords (esmvalcore.cmor.table.CMIP6Info attribute), 181  
 coords (esmvalcore.cmor.table.CustomInfo attribute), 186  
 coords (esmvalcore.cmor.table.InfoBase attribute), 188  
 coords (esmvalcore.cmor.table.NoInfo attribute), 189  
 coords (esmvalcore.cmor.table.Obs4MIPsInfo attribute), 191  
 copy() (esmvalcore.cmor.table.TableInfo method), 193  
 copy() (esmvalcore.cmor.table.VariableInfo method), 194  
 copy() (esmvalcore.config.Config method), 200  
 copy() (esmvalcore.config.Session method), 202  
 copy() (esmvalcore.dataset.Dataset method), 205  
 create() (esmvalcore.experimental.recipe\_output.DataFile class method), 295  
 create() (esmvalcore.experimental.recipe\_output.ImageFile class method), 296  
 create() (esmvalcore.experimental.recipe\_output.OutputFile class method), 297  
 create\_dataset\_map() (in module *esmvalcore.esgf.facets*), 214  
 create\_dataset\_map() (in module *esmvalcore.io.esgf.facets*), 225  
 cumulative\_sum() (in module *esmvalcore.preprocessor*), 253  
 CustomInfo (class in *esmvalcore.cmor.table*), 185
- ## D
- daily\_statistics() (in module *esmvalcore.preprocessor*), 253  
 data (esmvalcore.experimental.recipe.Recipe property), 290  
 data\_citation\_file (esmvalcore.experimental.recipe\_output.DataFile property), 295  
 data\_citation\_file (esmvalcore.experimental.recipe\_output.ImageFile property), 296  
 data\_citation\_file (esmvalcore.experimental.recipe\_output.OutputFile property), 297  
 data\_files (esmvalcore.experimental.recipe\_output.TaskOutput property), 299  
 DataElement (class in *esmvalcore.io.protocol*), 233  
 DataFile (class in *esmvalcore.experimental.recipe\_output*), 294  
 Dataset (class in *esmvalcore.dataset*), 204  
 dataset (esmvalcore.io.esgf.ESGFFile attribute), 219  
 DATASET\_MAP (in module *esmvalcore.io.esgf.facets*), 224  
 dataset\_to\_iris() (in module *esmvalcore.iris\_helpers*), 235  
 datasets\_to\_recipe() (in module *esmvalcore.dataset*), 207  
 DataSource (class in *esmvalcore.io.protocol*), 233  
 DataSource (class in *esmvalcore.local*), 238

- DataType (in module *esmvalcore.typing*), 288  
 date2num() (in module *esmvalcore.iris\_helpers*), 235  
 DEBUG (*esmvalcore.cmor.check.CheckLevels* attribute), 168  
 debug\_info (*esmvalcore.io.esgf.ESGFDataSource* attribute), 219  
 debug\_info (*esmvalcore.io.intake\_esgf.IntakeESGFDataSource* attribute), 228  
 debug\_info (*esmvalcore.io.local.LocalDataSource* attribute), 231  
 debug\_info (*esmvalcore.io.protocol.DataSource* attribute), 234  
 debug\_info (*esmvalcore.local.LocalDataSource* attribute), 240  
 decadal\_statistics() (in module *esmvalcore.preprocessor*), 254  
 DEFAULT (*esmvalcore.cmor.check.CheckLevels* attribute), 168  
 default (*esmvalcore.cmor.table.CMIP3Info* attribute), 177  
 default (*esmvalcore.cmor.table.CMIP5Info* attribute), 179  
 default (*esmvalcore.cmor.table.CMIP6Info* attribute), 181  
 default (*esmvalcore.cmor.table.CustomInfo* attribute), 186  
 default (*esmvalcore.cmor.table.InfoBase* attribute), 188  
 default (*esmvalcore.cmor.table.NoInfo* attribute), 189  
 default (*esmvalcore.cmor.table.Obs4MIPsInfo* attribute), 191  
 DEFAULT\_ORDER (in module *esmvalcore.preprocessor*), 242  
 default\_table\_prefix (*esmvalcore.cmor.table.CMIP6Info* attribute), 181  
 default\_table\_prefix (*esmvalcore.cmor.table.Obs4MIPsInfo* attribute), 191  
 depth\_integration() (in module *esmvalcore.preprocessor*), 254  
 derive() (in module *esmvalcore.preprocessor*), 254  
 detrend() (in module *esmvalcore.preprocessor*), 255  
 DiagnosticOutput (class in *esmvalcore.experimental.recipe\_output*), 295  
 diagnostics (*esmvalcore.experimental.recipe\_output.RecipeOutput* attribute), 298  
 dimensions (*esmvalcore.cmor.table.VariableInfo* attribute), 194  
 dirname\_template (*esmvalcore.io.local.LocalDataSource* attribute), 231  
 dirname\_template (*esmvalcore.local.LocalDataSource* attribute), 240  
 distance\_metric() (in module *esmvalcore.preprocessor*), 255  
 download() (*esmvalcore.io.esgf.ESGFFile* method), 220  
 download() (in module *esmvalcore.io.esgf*), 221  
 download\_dir (*esmvalcore.io.esgf.ESGFDataSource* attribute), 219
- ## E
- ensemble\_statistics() (in module *esmvalcore.preprocessor*), 256  
 Error, 215  
 ESGFDataSource (class in *esmvalcore.io.esgf*), 218  
 ESGFFile (class in *esmvalcore.io.esgf*), 219  
 esmvalcore.cmor module, 167  
 esmvalcore.cmor.check module, 167  
 esmvalcore.cmor.fix module, 172  
 esmvalcore.cmor.fixes module, 174  
 esmvalcore.cmor.table module, 175  
 esmvalcore.config module, 199  
 esmvalcore.dataset module, 204  
 esmvalcore.esgf.facets module, 214  
 esmvalcore.exceptions module, 214  
 esmvalcore.experimental.recipe module, 290  
 esmvalcore.experimental.recipe\_metadata module, 300  
 esmvalcore.experimental.recipe\_output module, 294  
 esmvalcore.experimental.utils module, 303  
 esmvalcore.io module, 216  
 esmvalcore.io.esgf module, 217  
 esmvalcore.io.esgf.facets module, 224  
 esmvalcore.io.intake\_esgf module, 225  
 esmvalcore.io.local module, 229  
 esmvalcore.io.protocol module, 232  
 esmvalcore.iris\_helpers module, 234  
 esmvalcore.local module, 238

- esmvalcore.preprocessor  
   module, 243
- esmvalcore.preprocessor.regrid\_schemes  
   module, 284
- esmvalcore.typing  
   module, 288
- ESMValCoreDeprecationWarning, 215
- ESMValCoreLoadWarning, 215
- ESMValCorePreprocessorWarning, 215
- ESMValCoreUserWarning, 215
- extract\_coordinate\_points() (in module *esmval-core.preprocessor*), 257
- extract\_levels() (in module *esmval-core.preprocessor*), 258
- extract\_location() (in module *esmval-core.preprocessor*), 258
- extract\_month() (in module *esmval-core.preprocessor*), 259
- extract\_named\_regions() (in module *esmval-core.preprocessor*), 259
- extract\_point() (in module *esmval-core.preprocessor*), 260
- extract\_region() (in module *esmval-core.preprocessor*), 261
- extract\_season() (in module *esmval-core.preprocessor*), 261
- extract\_shape() (in module *esmval-core.preprocessor*), 261
- extract\_surface\_from\_atm() (in module *esmval-core.preprocessor*), 262
- extract\_time() (in module *esmvalcore.preprocessor*), 262
- extract\_trajectory() (in module *esmval-core.preprocessor*), 263
- extract\_transect() (in module *esmval-core.preprocessor*), 264
- extract\_volume() (in module *esmval-core.preprocessor*), 264
- F**
- facets (*esmvalcore.dataset.Dataset* attribute), 204
- facets (*esmvalcore.io.esgf.ESGFFile* attribute), 219
- facets (*esmvalcore.io.intake\_esgf.IntakeESGFDataset* attribute), 229
- facets (*esmvalcore.io.intake\_esgf.IntakeESGFDataSource* attribute), 228
- facets (*esmvalcore.io.local.LocalFile* property), 232
- facets (*esmvalcore.io.protocol.DataElement* attribute), 233
- facets (*esmvalcore.local.LocalFile* property), 241
- FACETS (in module *esmvalcore.io.esgf.facets*), 224
- Facets (in module *esmvalcore.typing*), 288
- FacetValue (in module *esmvalcore.typing*), 288
- filename\_template (*esmval-core.io.local.LocalDataSource* attribute), 231
- filename\_template (*esmval-core.local.LocalDataSource* attribute), 240
- files (*esmvalcore.dataset.Dataset* property), 205
- FILTER\_ATTRS (*esmval-core.experimental.recipe\_output.RecipeOutput* attribute), 298
- find() (*esmvalcore.experimental.utils.RecipeList* method), 303
- find\_data() (*esmvalcore.io.esgf.ESGFDataSource* method), 219
- find\_data() (*esmval-core.io.intake\_esgf.IntakeESGFDataSource* method), 228
- find\_data() (*esmvalcore.io.local.LocalDataSource* method), 231
- find\_data() (*esmvalcore.io.protocol.DataSource* method), 234
- find\_data() (*esmvalcore.local.LocalDataSource* method), 240
- find\_files() (*esmvalcore.dataset.Dataset* method), 205
- find\_files() (*esmvalcore.local.DataSource* method), 239
- find\_files() (in module *esmvalcore.io.esgf*), 221
- find\_files() (in module *esmvalcore.local*), 241
- fix\_data() (in module *esmvalcore.cmor.fix*), 172
- fix\_data() (in module *esmvalcore.preprocessor*), 265
- fix\_file() (in module *esmvalcore.cmor.fix*), 173
- fix\_file() (in module *esmvalcore.preprocessor*), 265
- fix\_metadata() (in module *esmvalcore.cmor.fix*), 174
- fix\_metadata() (in module *esmvalcore.preprocessor*), 266
- frequency (*esmvalcore.cmor.check.CMORCheck* attribute), 168
- frequency (*esmvalcore.cmor.table.TableInfo* attribute), 193
- frequency (*esmvalcore.cmor.table.VariableInfo* attribute), 194
- from\_core\_recipe\_output() (*esmval-core.experimental.recipe\_output.RecipeOutput* class method), 298
- from\_dict() (*esmval-core.experimental.recipe\_metadata.Contributor* class method), 300
- from\_files() (*esmvalcore.dataset.Dataset* method), 205
- from\_ranges() (*esmvalcore.dataset.Dataset* method), 206
- from\_recipe() (*esmvalcore.dataset.Dataset* static method), 206
- from\_tag() (*esmvalcore.experimental.recipe\_metadata.Contributor*

- class method), 300  
 from\_tag() (*esmvalcore.experimental.recipe\_metadata.Project* class method), 301  
 from\_tag() (*esmvalcore.experimental.recipe\_metadata.Reference* class method), 301  
 from\_task() (*esmvalcore.experimental.recipe\_output.TaskOutput* class method), 299  
 fromkeys() (*esmvalcore.cmor.table.TableInfo* class method), 193
- ## G
- generic\_lev\_coords (*esmvalcore.cmor.table.CoordinateInfo* attribute), 184  
 generic\_lev\_name (*esmvalcore.cmor.table.CoordinateInfo* attribute), 184  
 GenericFuncScheme (class in *esmvalcore.preprocessor.regrid\_schemes*), 284  
 GenericRegridder (class in *esmvalcore.preprocessor.regrid\_schemes*), 284  
 get() (*esmvalcore.cmor.table.TableInfo* method), 193  
 get() (*esmvalcore.experimental.recipe\_output.RecipeOutput* method), 299  
 get\_all\_recipes() (in module *esmvalcore.experimental.utils*), 304  
 get\_glob\_patterns() (*esmvalcore.local.DataSource* method), 239  
 get\_next\_month() (in module *esmvalcore.cmor.fixes*), 175  
 get\_output() (*esmvalcore.experimental.recipe.Recipe* method), 290  
 get\_recipe() (in module *esmvalcore.experimental.utils*), 304  
 get\_table() (*esmvalcore.cmor.table.CMIP3Info* method), 177  
 get\_table() (*esmvalcore.cmor.table.CMIP5Info* method), 179  
 get\_table() (*esmvalcore.cmor.table.CMIP6Info* method), 181  
 get\_table() (*esmvalcore.cmor.table.CustomInfo* method), 186  
 get\_table() (*esmvalcore.cmor.table.InfoBase* method), 188  
 get\_table() (*esmvalcore.cmor.table.NoInfo* method), 189  
 get\_table() (*esmvalcore.cmor.table.Obs4MIPsInfo* method), 191  
 get\_tables() (in module *esmvalcore.cmor.table*), 195  
 get\_time\_bounds() (in module *esmvalcore.cmor.fixes*), 175  
 get\_var\_info() (in module *esmvalcore.cmor.table*), 196  
 get\_variable() (*esmvalcore.cmor.table.CMIP3Info* method), 177  
 get\_variable() (*esmvalcore.cmor.table.CMIP5Info* method), 179  
 get\_variable() (*esmvalcore.cmor.table.CMIP6Info* method), 181  
 get\_variable() (*esmvalcore.cmor.table.CustomInfo* method), 186  
 get\_variable() (*esmvalcore.cmor.table.InfoBase* method), 188  
 get\_variable() (*esmvalcore.cmor.table.NoInfo* method), 189  
 get\_variable() (*esmvalcore.cmor.table.Obs4MIPsInfo* method), 191  
 GRIB\_FORMATS (in module *esmvalcore.io.local*), 230
- ## H
- has\_coord\_with\_standard\_name() (*esmvalcore.cmor.table.VariableInfo* method), 194  
 has\_debug\_messages() (*esmvalcore.cmor.check.CMORCheck* method), 170  
 has\_errors() (*esmvalcore.cmor.check.CMORCheck* method), 170  
 has\_irregular\_grid() (in module *esmvalcore.iris\_helpers*), 236  
 has\_regular\_grid() (in module *esmvalcore.iris\_helpers*), 236  
 has\_unstructured\_grid() (in module *esmvalcore.iris\_helpers*), 236  
 has\_warnings() (*esmvalcore.cmor.check.CMORCheck* method), 170  
 histogram() (in module *esmvalcore.preprocessor*), 266  
 hourly\_statistics() (in module *esmvalcore.preprocessor*), 267
- ## I
- IGNORE (*esmvalcore.cmor.check.CheckLevels* attribute), 168  
 ignore\_iris\_vague\_metadata\_warnings() (in module *esmvalcore.iris\_helpers*), 236  
 ignore\_warnings (*esmvalcore.io.local.LocalDataSource* attribute), 231  
 ignore\_warnings (*esmvalcore.io.local.LocalFile* property), 232  
 ignore\_warnings (*esmvalcore.local.LocalDataSource* attribute), 240  
 ignore\_warnings (*esmvalcore.local.LocalFile* property), 241  
 ignore\_warnings\_context() (in module *esmvalcore.iris\_helpers*), 236  
 image\_files (*esmvalcore.experimental.recipe\_output.TaskOutput* property), 300

- ImageFile (class in *esmvalcore.experimental.recipe\_output*), 295
- info (*esmvalcore.experimental.recipe\_output.RecipeOutput* attribute), 298
- InfoBase (class in *esmvalcore.cmor.table*), 187
- INHERITED\_FACETS (in module *esmvalcore.dataset*), 207
- InputFilesNotFound, 215
- institutes (*esmvalcore.cmor.table.CMIP6Info* attribute), 182
- institutes (*esmvalcore.cmor.table.Obs4MIPsInfo* attribute), 192
- IntakeESGFDataset (class in *esmvalcore.io.intake\_esgf*), 228
- IntakeESGFDataSource (class in *esmvalcore.io.intake\_esgf*), 227
- InvalidConfigParameter, 215
- IrisESMFRegrid (class in *esmvalcore.preprocessor.regrid\_schemes*), 284
- items() (*esmvalcore.cmor.table.TableInfo* method), 193
- items() (*esmvalcore.experimental.recipe\_output.RecipeOutput* method), 299
- ## J
- JsonInfo (class in *esmvalcore.cmor.table*), 188
- ## K
- keys() (*esmvalcore.cmor.table.TableInfo* method), 193
- keys() (*esmvalcore.experimental.recipe\_output.RecipeOutput* method), 299
- kind (*esmvalcore.experimental.recipe\_output.DataFile* attribute), 295
- kind (*esmvalcore.experimental.recipe\_output.ImageFile* attribute), 296
- kind (*esmvalcore.experimental.recipe\_output.OutputFile* attribute), 297
- kwargs (*esmvalcore.preprocessor.regrid\_schemes.IrisESMFRegrid* attribute), 286
- ## L
- linear\_trend() (in module *esmvalcore.preprocessor*), 268
- linear\_trend\_stderr() (in module *esmvalcore.preprocessor*), 268
- load() (*esmvalcore.dataset.Dataset* method), 206
- load() (in module *esmvalcore.preprocessor*), 268
- load\_data\_sources() (in module *esmvalcore.io*), 217
- load\_from\_dirs() (*esmvalcore.config.Config* method), 200
- load\_iris() (*esmvalcore.experimental.recipe\_output.DataFile* method), 295
- load\_xarray() (*esmvalcore.experimental.recipe\_output.DataFile* method), 295
- local\_file() (*esmvalcore.io.esgf.ESGFFile* method), 220
- local\_solar\_time() (in module *esmvalcore.preprocessor*), 269
- LocalDataSource (class in *esmvalcore.io.local*), 230
- LocalDataSource (class in *esmvalcore.local*), 239
- LocalFile (class in *esmvalcore.io.local*), 232
- LocalFile (class in *esmvalcore.local*), 240
- long\_name (*esmvalcore.cmor.table.CoordinateInfo* attribute), 184
- long\_name (*esmvalcore.cmor.table.VariableInfo* attribute), 195
- ## M
- main\_log (*esmvalcore.config.Session* property), 203
- main\_log\_debug (*esmvalcore.config.Session* property), 203
- mask\_above\_threshold() (in module *esmvalcore.preprocessor*), 270
- mask\_below\_threshold() (in module *esmvalcore.preprocessor*), 270
- mask\_fillvalues() (in module *esmvalcore.preprocessor*), 270
- mask\_glaciated() (in module *esmvalcore.preprocessor*), 271
- mask\_inside\_range() (in module *esmvalcore.preprocessor*), 271
- mask\_landsea() (in module *esmvalcore.preprocessor*), 271
- mask\_landseaice() (in module *esmvalcore.preprocessor*), 272
- mask\_multimodel() (in module *esmvalcore.preprocessor*), 272
- mask\_outside\_range() (in module *esmvalcore.preprocessor*), 273
- mask\_page\_attributes() (in module *esmvalcore.iris\_helpers*), 237
- meridional\_statistics() (in module *esmvalcore.preprocessor*), 273
- minimal\_facets (*esmvalcore.dataset.Dataset* property), 207
- MissingConfigParameter, 215
- modeling\_realm (*esmvalcore.cmor.table.VariableInfo* attribute), 195
- module
- esmvalcore.cmor*, 167
  - esmvalcore.cmor.check*, 167
  - esmvalcore.cmor.fix*, 172
  - esmvalcore.cmor.fixes*, 174
  - esmvalcore.cmor.table*, 175
  - esmvalcore.config*, 199
  - esmvalcore.dataset*, 204
  - esmvalcore.esgf.facets*, 214
  - esmvalcore.exceptions*, 214

- esmvalcore.experimental.recipe, 290  
 esmvalcore.experimental.recipe\_metadata, 300  
 esmvalcore.experimental.recipe\_output, 294  
 esmvalcore.experimental.utils, 303  
 esmvalcore.io, 216  
 esmvalcore.io.esgf, 217  
 esmvalcore.io.esgf.facets, 224  
 esmvalcore.io.intake\_esgf, 225  
 esmvalcore.io.local, 229  
 esmvalcore.io.protocol, 232  
 esmvalcore.iris\_helpers, 234  
 esmvalcore.local, 238  
 esmvalcore.preprocessor, 243  
 esmvalcore.preprocessor.regrid\_schemes, 284  
 esmvalcore.typing, 288  
 monthly\_statistics() (in module *esmvalcore.preprocessor*), 273  
 multi\_model\_statistics() (in module *esmvalcore.preprocessor*), 274  
 must\_have\_bounds (esmvalcore.cmor.table.CoordinateInfo attribute), 185
- ## N
- name (esmvalcore.cmor.table.CoordinateInfo attribute), 185  
 name (esmvalcore.cmor.table.TableInfo attribute), 193  
 name (esmvalcore.experimental.recipe.Recipe property), 290  
 name (esmvalcore.io.esgf.ESGFDataSource attribute), 219  
 name (esmvalcore.io.esgf.ESGFFile attribute), 219  
 name (esmvalcore.io.intake\_esgf.IntakeESGFDataset attribute), 229  
 name (esmvalcore.io.intake\_esgf.IntakeESGFDataSource attribute), 228  
 name (esmvalcore.io.local.LocalDataSource attribute), 231  
 name (esmvalcore.io.protocol.DataElement attribute), 233  
 name (esmvalcore.io.protocol.DataSource attribute), 234  
 name (esmvalcore.local.LocalDataSource attribute), 240  
 nested\_update() (esmvalcore.config.Config method), 200  
 NetCDFAttr (in module *esmvalcore.typing*), 288  
 NoInfo (class in *esmvalcore.cmor.table*), 189
- ## O
- Obs4MIPsInfo (class in *esmvalcore.cmor.table*), 190  
 out\_name (esmvalcore.cmor.table.CoordinateInfo attribute), 185
- OutputFile (class in *esmvalcore.experimental.recipe\_output*), 296
- ## P
- path2facets() (*esmvalcore.local.DataSource* method), 239  
 paths (*esmvalcore.cmor.table.CMIP3Info* attribute), 178  
 paths (*esmvalcore.cmor.table.CMIP5Info* attribute), 180  
 paths (*esmvalcore.cmor.table.CMIP6Info* attribute), 182  
 paths (*esmvalcore.cmor.table.CustomInfo* attribute), 187  
 paths (*esmvalcore.cmor.table.InfoBase* attribute), 188  
 paths (*esmvalcore.cmor.table.NoInfo* attribute), 190  
 paths (*esmvalcore.cmor.table.Obs4MIPsInfo* attribute), 192  
 plot\_dir (*esmvalcore.config.Session* property), 203  
 pop() (*esmvalcore.cmor.table.TableInfo* method), 193  
 popitem() (*esmvalcore.cmor.table.TableInfo* method), 193  
 positive (*esmvalcore.cmor.table.VariableInfo* attribute), 195  
 prepare() (*esmvalcore.io.esgf.ESGFFile* method), 220  
 prepare() (*esmvalcore.io.intake\_esgf.IntakeESGFDataset* method), 229  
 prepare() (*esmvalcore.io.local.LocalFile* method), 232  
 prepare() (*esmvalcore.io.protocol.DataElement* method), 233  
 prepare() (*esmvalcore.local.LocalFile* method), 241  
 preproc\_dir (*esmvalcore.config.Session* property), 203  
 priority (*esmvalcore.io.esgf.ESGFDataSource* attribute), 219  
 priority (*esmvalcore.io.intake\_esgf.IntakeESGFDataSource* attribute), 228  
 priority (*esmvalcore.io.local.LocalDataSource* attribute), 231  
 priority (*esmvalcore.io.protocol.DataSource* attribute), 234  
 priority (*esmvalcore.local.LocalDataSource* attribute), 240  
 Project (class in *esmvalcore.experimental.recipe\_metadata*), 301  
 project (*esmvalcore.io.esgf.ESGFDataSource* attribute), 219  
 project (*esmvalcore.io.intake\_esgf.IntakeESGFDataSource* attribute), 228  
 project (*esmvalcore.io.local.LocalDataSource* attribute), 231  
 project (*esmvalcore.io.protocol.DataSource* attribute), 234  
 project (*esmvalcore.local.LocalDataSource* attribute), 240  
 provenance\_xml\_file (*esmvalcore.experimental.recipe\_output.DataFile* property), 295

- `provenance_xml_file` (*esmval-core.experimental.recipe\_output.ImageFile* property), 296  
`provenance_xml_file` (*esmval-core.experimental.recipe\_output.OutputFile* property), 297
- ## R
- `read_cmor_tables()` (in module *esmval-core.cmor.table*), 196  
`read_json()` (*esmvalcore.cmor.table.CoordinateInfo* method), 185  
`read_json()` (*esmvalcore.cmor.table.VariableInfo* method), 195  
`read_main_log()` (*esmval-core.experimental.recipe\_output.RecipeOutput* method), 299  
`read_main_log_debug()` (*esmval-core.experimental.recipe\_output.RecipeOutput* method), 299  
`realm` (*esmvalcore.cmor.table.TableInfo* attribute), 193  
`rechunk_cube()` (in module *esmvalcore.iris\_helpers*), 237  
`Recipe` (class in *esmvalcore.experimental.recipe*), 290  
`RecipeError`, 215  
`RecipeList` (class in *esmvalcore.experimental.utils*), 303  
`RecipeOutput` (class in *esmval-core.experimental.recipe\_output*), 298  
`Reference` (class in *esmval-core.experimental.recipe\_metadata*), 301  
`references` (*esmvalcore.experimental.recipe\_output.DataFile* property), 295  
`references` (*esmvalcore.experimental.recipe\_output.ImageFile* property), 296  
`references` (*esmvalcore.experimental.recipe\_output.OutputFile* property), 297  
`regex_pattern` (*esmvalcore.local.DataSource* property), 239  
`regrid()` (in module *esmvalcore.preprocessor*), 275  
`regrid_time()` (in module *esmvalcore.preprocessor*), 277  
`regridder()` (*esmval-core.preprocessor.regrid\_schemes.GenericFuncScheme* method), 284  
`regridder()` (*esmval-core.preprocessor.regrid\_schemes.IrisESMFRegridder* method), 286  
`regridder()` (*esmval-core.preprocessor.regrid\_schemes.UnstructuredLinear* method), 287  
`regridder()` (*esmval-core.preprocessor.regrid\_schemes.UnstructuredNetCDF* method), 287  
`relative_cmor_log` (*esmvalcore.config.Session* attribute), 203  
`relative_main_log` (*esmvalcore.config.Session* attribute), 203  
`relative_main_log_debug` (*esmval-core.config.Session* attribute), 203  
`relative_plot_dir` (*esmvalcore.config.Session* attribute), 203  
`relative_preproc_dir` (*esmvalcore.config.Session* attribute), 203  
`relative_run_dir` (*esmvalcore.config.Session* attribute), 203  
`relative_work_dir` (*esmvalcore.config.Session* attribute), 203  
`RELAXED` (*esmvalcore.cmor.check.CheckLevels* attribute), 168  
`reload()` (*esmvalcore.config.Config* method), 200  
`remove_supplementary_variables()` (in module *esmvalcore.preprocessor*), 278  
`render()` (*esmvalcore.experimental.recipe.Recipe* method), 290  
`render()` (*esmvalcore.experimental.recipe\_metadata.Reference* method), 301  
`render()` (*esmvalcore.experimental.recipe\_output.RecipeOutput* method), 299  
`RenderError`, 302  
`report()` (*esmvalcore.cmor.check.CMORCheck* method), 170  
`report_critical()` (*esmval-core.cmor.check.CMORCheck* method), 170  
`report_debug_message()` (*esmval-core.cmor.check.CMORCheck* method), 171  
`report_debug_messages()` (*esmval-core.cmor.check.CMORCheck* method), 170  
`report_error()` (*esmvalcore.cmor.check.CMORCheck* method), 170  
`report_errors()` (*esmval-core.cmor.check.CMORCheck* method), 169  
`report_warning()` (*esmval-core.cmor.check.CMORCheck* method), 171  
`report_warnings()` (*esmval-core.cmor.check.CMORCheck* method), 169  
`requested` (*esmvalcore.cmor.table.CoordinateInfo* attribute), 185  
`resample_hours()` (in module *esmval-core.preprocessor*), 278  
`resample_time()` (in module *esmval-core.preprocessor*), 279  
`rolling_window_statistics()` (in module *esmval-core.preprocessor*), 279  
`rootpath` (*esmvalcore.io.local.LocalDataSource* attribute), 231  
`rootpath` (*esmvalcore.local.LocalDataSource* attribute), 240

- `run()` (*esmvalcore.experimental.recipe.Recipe* method), 290
- `run_dir` (*esmvalcore.config.Session* property), 203
- ## S
- `safe_convert_units()` (in module *esmvalcore.iris\_helpers*), 237
- `save()` (in module *esmvalcore.preprocessor*), 280
- `seasonal_statistics()` (in module *esmvalcore.preprocessor*), 280
- `Session` (class in *esmvalcore.config*), 201
- `session` (*esmvalcore.dataset.Dataset* property), 207
- `session` (*esmvalcore.experimental.recipe\_output.RecipeOutput* attribute), 298
- `session_dir` (*esmvalcore.config.Session* property), 203
- `session_name` (*esmvalcore.config.Session* attribute), 203
- `set_facet()` (*esmvalcore.dataset.Dataset* method), 207
- `set_session_name()` (*esmvalcore.config.Session* method), 203
- `set_version()` (*esmvalcore.dataset.Dataset* method), 207
- `setdefault()` (*esmvalcore.cmor.table.TableInfo* method), 193
- `short_name` (*esmvalcore.cmor.table.VariableInfo* attribute), 195
- `size` (*esmvalcore.io.esgf.ESGFFFile* attribute), 219
- `standard_name` (*esmvalcore.cmor.table.CoordinateInfo* attribute), 185
- `standard_name` (*esmvalcore.cmor.table.VariableInfo* attribute), 195
- `start_session()` (*esmvalcore.config.Config* method), 201
- `stored_direction` (*esmvalcore.cmor.table.CoordinateInfo* attribute), 185
- `STRICT` (*esmvalcore.cmor.check.CheckLevels* attribute), 168
- `strict` (*esmvalcore.cmor.table.CMIP3Info* attribute), 178
- `strict` (*esmvalcore.cmor.table.CMIP5Info* attribute), 180
- `strict` (*esmvalcore.cmor.table.CMIP6Info* attribute), 182
- `strict` (*esmvalcore.cmor.table.CustomInfo* attribute), 187
- `strict` (*esmvalcore.cmor.table.InfoBase* attribute), 188
- `strict` (*esmvalcore.cmor.table.NoInfo* attribute), 190
- `strict` (*esmvalcore.cmor.table.Obs4MIPsInfo* attribute), 192
- `summary()` (*esmvalcore.dataset.Dataset* method), 207
- `supplementaries` (*esmvalcore.dataset.Dataset* attribute), 204
- `SuppressedError`, 215
- ## T
- `TableInfo` (class in *esmvalcore.cmor.table*), 192
- `tables` (*esmvalcore.cmor.table.CMIP3Info* attribute), 178
- `tables` (*esmvalcore.cmor.table.CMIP5Info* attribute), 180
- `tables` (*esmvalcore.cmor.table.CMIP6Info* attribute), 182
- `tables` (*esmvalcore.cmor.table.CustomInfo* attribute), 187
- `tables` (*esmvalcore.cmor.table.InfoBase* attribute), 188
- `tables` (*esmvalcore.cmor.table.NoInfo* attribute), 190
- `tables` (*esmvalcore.cmor.table.Obs4MIPsInfo* attribute), 192
- `TaskOutput` (class in *esmvalcore.experimental.recipe\_output*), 299
- `timeseries_filter()` (in module *esmvalcore.preprocessor*), 281
- `to_base64()` (*esmvalcore.experimental.recipe\_output.ImageFile* method), 296
- `to_iris()` (*esmvalcore.io.esgf.ESGFFFile* method), 220
- `to_iris()` (*esmvalcore.io.intake\_esgf.IntakeESGFFDataset* method), 229
- `to_iris()` (*esmvalcore.io.local.LocalFile* method), 232
- `to_iris()` (*esmvalcore.io.protocol.DataElement* method), 233
- `to_iris()` (*esmvalcore.local.LocalFile* method), 241
- ## U
- `units` (*esmvalcore.cmor.table.CoordinateInfo* attribute), 185
- `units` (*esmvalcore.cmor.table.VariableInfo* attribute), 195
- `UnstructuredLinear` (class in *esmvalcore.preprocessor.regrid\_schemes*), 286
- `UnstructuredLinearRegridder` (class in *esmvalcore.preprocessor.regrid\_schemes*), 287
- `UnstructuredNearest` (class in *esmvalcore.preprocessor.regrid\_schemes*), 287
- `update()` (*esmvalcore.cmor.table.TableInfo* method), 193
- `update_from_dirs()` (*esmvalcore.config.Config* method), 201
- `urls` (*esmvalcore.io.esgf.ESGFFFile* attribute), 220
- ## V
- `valid_max` (*esmvalcore.cmor.table.CoordinateInfo* attribute), 185
- `valid_max` (*esmvalcore.cmor.table.VariableInfo* attribute), 195
- `valid_min` (*esmvalcore.cmor.table.CoordinateInfo* attribute), 185

*valid\_min* (*esmvalcore.cmor.table.VariableInfo* attribute), 195  
*value* (*esmvalcore.cmor.table.CoordinateInfo* attribute), 185  
*values* (*esmvalcore.io.intake\_esgf.IntakeESGFDataSource* attribute), 228  
*values()* (*esmvalcore.cmor.table.TableInfo* method), 193  
*values()* (*esmvalcore.experimental.recipe\_output.RecipeOutput* method), 299  
*var\_name* (*esmvalcore.cmor.table.CoordinateInfo* attribute), 185  
*var\_to\_freq* (*esmvalcore.cmor.table.CMIP6Info* attribute), 182  
*var\_to\_freq* (*esmvalcore.cmor.table.CustomInfo* attribute), 187  
*var\_to\_freq* (*esmvalcore.cmor.table.Obs4MIPsInfo* attribute), 192  
*VariableInfo* (class in *esmvalcore.cmor.table*), 193  
*volume\_statistics()* (in module *esmvalcore.preprocessor*), 281

## W

*weighting\_landsea\_fraction()* (in module *esmvalcore.preprocessor*), 282  
*with\_traceback()* (*esmvalcore.experimental.recipe\_metadata.RenderError* method), 302  
*work\_dir* (*esmvalcore.config.Session* property), 203  
*write\_html()* (*esmvalcore.experimental.recipe\_output.RecipeOutput* method), 299

## Z

*zonal\_statistics()* (in module *esmvalcore.preprocessor*), 283