Earth Diagnostics Documentation

Release 3.0.0

BSC-CNS Earth Sciences Department

Aug 23, 2018

Contents

CHAPTER 1

Tutorial

So, you are planning to use the Earth Diagnostics? You don't know how to use them? This is the place to go. From now on this tutorial will guide you through all the process from installation to running.

Hint: If you have any problem with this tutorial, please report it to <javier.vegas@bsc.es> so it can be corrected. A lof of people will benefit from it.

1.1 Installation

If you have access to the BSC-ES machines, you don't need to install it. Just use the available module:

In case that you need a custom installation for development or can not use the BSC-ES machines, install it from BSC-ES GitLab repository:

```
pip install git+https://earth.bsc.es/gitlab/es/ocean_diagnostics.git
```

You will also need

- CDO version 1.7.2 (other versions could work, but this is the one we use)
- NCO version 4.5.4 or newer
- Python 2.7 or newer (but no 3.x) with bscearth.utils, CDO and NCO packages, among others.
- Access to CDFTOOLS_3.0 executables for BSC-ES. The source code is available on Github (https://github.com/jvegasbsc/CDFTOOLS) and it can be compiled with CMake

1.2 Creating a config file

Go to the folder where you installed the EarthDiagnostics. You will see a folder called earthdiagnostics, and, inside it, the model_diags.conf file that can be used as a template for your config file. Create a copy of it wherever it suites you.

Now open your brand new copy with your preferred text editor. The file contains commentaries explaining each one of its options, so read it carefully and edit whatever you need. Don't worry about DIAGS option, we will talk about it next.

After this, you need to choose the diagnostics you want to run. For a simple test, it's recommended to use the monmean diagnostic to compute monthly means from daily data. We recommend it because it can be used with any variable, the user has to provide parameters but they are quite intuitive and it's relatively fast to compute. If your experiment does not have daily data, you can use any other diagnostic. Check next section for a list of available diagnostics and choose whichever suits you better. From now on, we will assume that you are going to run the monmean diagnostic.

Hint: For old Ocean Diagnostics users: you can use most of the old names as aliases to launch one or multiple diagnostics. Check the ALIAS section on the model_diags.conf to see which ones are available.

First, choose a variable that has daily data. Then replace the DIAGS option with the next one where $VARIABLE represents the variable's name and $DOMAIN its domain (atmos, ocean, seaice, landice...)

```
DIAGS = monmean,$DOMAIN,$VARIABLE
```

1.3 Prepare the run script

Once you have configured your experiment you can execute any diagnostic with the provided model_launch_diags.sh script. Create a copy and change the variable PATH_TO_CONF_FILE so it points to your conf file .

Now, execute the script (or submit it to bsceslogin01, it has the correct header) and... that's it! You will find your results directly on the storage and a folder for the temp files in the scratch named after the EXPID.

CHAPTER 2

Configuration file options

This section contains the list and explanation about all the options that are available on the configuration file. Use it as a reference while preparing your configuration file. Each subsection will refer to the matching section from the config file. Those subsections explanation may be divided itself for the shake of clarity but this further divisions have nothing to do with the config file syntax itself.

2.1 DIAGNOSTICS

This section contains the general configuration for the diagnostics. The explanation has been divided in two subsections: the first one will cover all the mandatory options that you must specify in every configuration, while the second will cover all the optional configurations.

2.1.1 Mandatory configurations

- **SCRATCH_DIR:** Temporary folder for the calculations. Final results will never be stored here.
- **DATA_DIR:** ':' separated list of folders to look for data in. It will look for file in the path $DATA_FOLDER/$EXPID and $DATA_FOLDER/$DATA_TYPE/$MODEL/$EXPID
- **CON_FILES:** Folder containing mask and mesh files for the dataset.
- **FREQUENCY:** Default data frequency to be used by the diagnostics. Some diagnostics can override this configuration or even ignore it completely.
- **DIAGS:** List of diagnostic to run. No specific order is needed: data dependencies will be enforced.

2.1.2 Optional configurations

- **SCRATCH_MASKS** Common scratch folder for the ocean masks. This is useful to avoid replicating them for each run at the fat nodes. By default is '/scratch/Earth/ocean_masks'

- **RESTORE_MESHES** By default, Earth Diagnostics only copies the mask files if they are not present in the scratch folder. If this option is set to true, Earth Diagnostics will copy them regardless of existence. Default is False.
- **DATA_ADAPTOR** This is used to choose the mechanism for storing and retrieving data. Options are CMOR (for our own experiments) or THREDDS (for anything else). Default value is CMOR
- **DATA_TYPE** Type of the dataset to use. It can be exp, obs or recon. Default is exp.
- **DATA_CONVENTION** Convention to use for file paths and names and variable naming among other things. Can be SPECS, PREFACE, PRIMAVERA or CMIP6. Default is SPECS.
- **CDFTOOLS_PATH** Path to the folder containing CDFTOOLS executables. By default is empty, so CDFTOOLS binaries must be added to the system path.
- **MAX_CORES** Maximum number of cores to use. By default the diagnostics will use all cores available to them. It is not necessary when launching through a scheduler, as Earthdiagnostics can detect how many cores the scheduler has allocated to it.
- **AUTO_CLEAN** If True, EarthDiagnostics removes the temporary folder just after finsihing. If RAM_DISK is set to True, this value is ignored and always Default is True
- **RAM_DISK** If set to True, the temporary files is created at the /dev/shm partition. This partition is not mounted from a disk. Instead, all files are created in the RAM memory, so hopefully this will improve performance at the cost of a much higher RAM consumption. Default is False.
- **MESH_MASK** Custom file to use instead of the corresponding mesh mask file.
- **NEW_MASK_GLO** Custom file to use instead of the corresponding new mask glo file
- **MASK_REGIONS** Custom file to use instead of the corresponding 2D regions file
- **MASK_REGIONS_3D** Custom file to use instead of the corresponding 3D regions file

2.2 EXPERIMENT

This sections contains options related to the experiment's definition or configuration.

- **MODEL** Name of the model used for the experiment.
- **MODEL_VERSION** Model version. Used to get the correct mask and mesh files
- **ATMOS_TIMESTEP** Time between outputs from the atmosphere. This is not the model simulation timestep! Default is 6.
- **OCEAN_TIMESTEP** Time between outputs from the ocean. This is not the model simulation timestep! Default is 6.
- **ATMOS_GRID** Atmospheric grid definition. Will be used as a default target for interpolation diagnostics.
- **INSTITUTE** Institute that made the experiment, observation or reconstruction
- **EXPID** Unique identifier for the experiment
- **NAME** Experiment's name. By default it is the EXPID.
- **STARTDATES** Startdates to run as a space separated list
- **MEMBER** Members to run as a space separated list. You can just provide the number or also add the prefix
- **MEMBER_DIGITS** Number of minimum digits to compose the member name. By default it is 1. For example, for member 1 member name will be fc1 if MEMBER_DIGITS is 1 or fc01 if MEMBER_DIGITS is 2

- **MEMBER_PREFIX** Prefix to use for the member names. By default is 'fc'
- **MEMBER_COUNT_START** Number corresponding to the first member. For example, if your first member is 'fc1', it should be 1. If it is 'fc0', it should be 0. By default is 0
- **CHUNK_SIZE** Length of the chunks in months
- **CHUNKS** Number of chunks to run
- **CHUNK_LIST** List of chunks to run. If empty, all diagnostics will be applied to all chunks
- **CALENDAR** Calendar to use for date calculation. All calendars supported by Autosubmit are available. Default is 'standard'

2.3 CMOR

In this section, you can control how will work the cmorization process. All options belonging to this section are optional.

2.3.1 Cmorization options

This options control when and which varibales will be cmorized.

- **FORCE** If True, launches the cmorization, regardless of existence of the extracted files or the package containing the online-cmorized ones. If False, only the non-present chunks will be cmorized. Default value is False
- **FORCE_UNTAR** Unpacks the online-cmorized files regardless of exstience of extracted files. If 'FORCE is True, this parameter has no effect. If False, only the non-present chunks will be unpacked. Default value is False.
- **FILTER_FILES** Only cmorize original files containing any of the given strings. This is a space separated list. Default is the empty string.
- **OCEAN_FILES** Boolean flag to activate or no NEMO files cmorization. Default is True.
- **ATMOSPHERE_FILES** Boolean flag to activate or no IFS files cmorization. Default is True.
- **USE_GRIB** Boolean flag to activate or no GRIB files cmorization for the atmosphere. If activated and no GRIB files are present, it will cmorize using the MMA files instead (as if it was set to False). Default is True.
- **CHUNKS** Space separated list of chunks to be cmorized. If not provided, all chunks are cmorized
- **VARIABLE_LIST** Space separated list of variables to cmorize. Variables must be specified as domain:var_name. If no one is specified, all the variables will be cmorized

Grib variables extraction

These three options ares used to configure the variables to be CMORized from the grib atmospheric files. They must be specified using the IFS code in a list separated by comma.

You can also specify the levels to extract using one of the the following syntaxes:

- VARIABLE_CODE
- VARIABLE_CODE:LEVEL,
- VARIABLE_CODE:LEVEL_1-LEVEL_2-...-LEVEL_N

- VARIABLE_CODE:MIN_LEVEL:MAX_LEVEL:STEP

Some examples to clarify it further: * Variable with code 129 at level 30000: 129:30000 * Variable with code 129 at levels 30000, 40000 and 60000: 129:30000-40000-60000 * Variable with code 129 at levels between 30000 and 600000 with 10000 intervals:

129:30000:60000:10000 equivalent to 129:30000-40000-50000-60000

- **ATMOS_HOURLY_VARS** Configuration of variables to be extracted in an hourly basis
- **ATMOS_DAILY_VARS** Configuration of variables to be extracted in a daily basis
- **ATMOS_MONTHLY_VARS** Configuration of variables to be extracted in a monthly basis

2.3.2 Metadata options

All the options in this subsection will serve just to add the given values to the homonymous attributes in the cmorized files.

- **ASSOCIATED_EXPERIMENT** Default value is 'to be filled'
- **ASSOCIATED_MODEL** Default value is 'to be filled'
- **INITIALIZATION_DESCRIPTION** Default value is 'to be filled'
- **INITIALIZATION_METHOD** Default value is '1'
- **PHYSICS_DESCRIPTION** Default value is 'to be filled'
- **PHYSICS_VERSION** Default value is '1'
- **SOURCE** Default value is 'to be filled'
- **VERSION** Dataset version to use (not present in all conventions)
- **DEFAULT_OCEAN_GRID** Name of the default ocean grid for those conventions that require it (CMIP6 and PRIMAVERA). Default is gn.
- **DEFAULT_ATMOS_GRID** Name of the default atmos grid for those conventions that require it (CMIP6 and PRIMAVERA). Default is gr.
- **ACTIVITY** Name of the activity. Default is CMIP

2.4 THREDDS

For now, there is only one option for the THREDDS server configuration.

- **SERVER_URL** THREDDS server URL

2.5 ALIAS

This config file section is different from all the others because it does not contain a set of configurations. Instead, in this section the user can define a set of aliases to be able to launch its most used configurations with ease. To do this, the user must add an option with named after the desired alias and assign to it the configuration or configurations to launch when this ALIAS is invoked. See the next example:

```
ALIAS_NAME = diag,opt1,opt2 diag,opt1new,opt2
```

In this case, the user has defined a new alias 'ALIAS' that can be used launch two times the diagnostic 'diag', the first with the options 'opt1' and 'opt2' and the second replacing 'opt1' with 'opt1new'.

In this example, configuring the DIAGS as

```
DIAGS = ALIAS_NAME
```

will be identical to

```
DIAGS = diag,opt1,opt2 diag,opt1new,opt2
```

coding=utf-8

CHAPTER 3

Diagnostic list

In this section you have a list of the available diagnostics, with a small description of each one and a link to the full documentation. To see what options are available for each diagnostic, see generate_jobs documentation.

Remember that diagnostics are specified separated by spaces while options are given separated by commas:

```
DIAGS = diag1 diag2,option1,option2 diag3
```

3.1 General

The diagnostics from this section are of general use and can be used with any variable you may have. Most of them are meant to help you to solve usual issues that you may have with the data: incorrect metadata, scaled up or down variables, links missing. This section also contains the diagnostic used to calculate the monthly means.

3.1.1 att

Writes a global attributte to all the netCDF files for a given variable. See *Attribute*

Options:

1. **Variable:** Variable name
2. **Domain:** Variable domain
3. **Attributte name:** Attributte to write
4. **Attribute value:** Atrribute's new value. Replace ',' with '&;' and ' ' with '&.' to avoid parsing errors when processing the diags
5. **Grid = '':** Variable grid. Only required in case that you want to use interpolated data.

3.1.2 dailymean

Calculates the daily mean for a given variable. See `DailyMean`

> **Warning:** This diagnostic does not use the frequency configuration from the config file. You must specify the original frequency when calling it.

Options:

1. **Variable:** Variable name
2. **Domain:** Variable domain
3. **Original frequency:** Original frequency to use
4. **Grid = '':** Variable grid. Only required in case that you want to use interpolated data.

3.1.3 module

Calculates the module for two given variables and stores the result in a third. See *`Module`*

Options:

1. **Domain:** Variables domain
2. **Variable U:** Variable U name
3. **Variable V:** Variable V name
4. **Variable Module:** Variable module name
5. **Grid = '':** Variable grids. Only required in case that you want to use interpolated data.

3.1.4 monmean

Calculates the monthly mean for a given variable. See `MonthlyMean`

> **Warning:** This diagnostic does not use the frequency configuration from the config file. You must specify the original frequency when calling it. Otherwise, it will always try to use daily data.

Options:

1. **Variable:** Variable name
2. **Domain:** Variable domain
3. **Original frequency = daily:** Original frequency to use
4. **Grid = '':** Variable grid. Only required in case that you want to use interpolated data.

3.1.5 relink

Regenerates the links created in the monthly_mean, daily_mean, etc folders for a given varible. See *Relink*

Options:

1. **Variable:** Variable name
2. **Domain:** Variable domain
3. **Move old =** True: If True, any data founded in the target directory will be moved to another folder (called FOLDER_NAME_old) instead of deleted.
4. **Grid = '':** Variable grid. Only required in case that you want to use interpolated data.

3.1.6 relinkall

Regenerates the links created in the monthly_mean, daily_mean, etc folders for all variables See *RelinkAll*

Options:

This diagnostic has no options

3.1.7 rewrite:

Just rewrites the CMOR output of a given variable. Useful to correct metadata or variable units. See *Rewrite*

Options:

1. **Variable:** Variable name
2. **Domain:** Variable domain
3. **Grid = '':** Variable grid. Only required in case that you want to use interpolated data.

3.1.8 scale

Scales a given variable using a given scale factor and offset (NEW_VALUE = OLD_VALUE * scale + offset). Useful to correct errors on the data.

See *Scale*

Options:

1. **Variable:** Variable name
2. **Domain:** Variable domain
3. **Scale value:** Scale factor for the variable
4. **Offset value:** Value to add to the original value after scaling
5. **Grid = '':** Variable grid. Only required in case that you want to use interpolated data.

6. **Min limit = NaN:** If there is any value below this threshold, scale will not be applied
7. **Max limit = NaN:** If there is any value above this threshold, scale will not be applied
8. **Frequencies = [*Default_frequency*]:** List of frequencies ('-' separated) to apply the scale on. Default is the frequency defined globally for all the diagnostics

3.1.9 simdim

Convert i j files to lon lat when there is no interpolation required, i.e. lon is constant over i and lat is constat over j

See *SimplifyDimensions*

Options:

1. **Domain:** Variable domain
2. **Variable:** Variable name
5. **Grid = '':** Variable grid. Only required in case that you want to use interpolated data.

3.1.10 yearlymean

Calculates the daily mean for a given variable. See `YearlyMean`

> **Warning:** This diagnostic does not use the frequency configuration from the config file. You must specify the original frequency when calling it.

Options:

1. **Variable:** Variable name
2. **Domain:** Variable domain
3. **Original frequency:** Original frequency to use
4. **Grid = '':** Variable grid. Only required in case that you want to use interpolated data.

3.2 Ocean

The diagnostics from this section are meant to be used with NEMO variables. Some of them will compute new variables while others just calculate means or sections for variables in the ORCA grid. The interpolation diagnostics are also included here as they are usually used with variables in the ORCA grid.

3.2.1 areamoc

Compute an Atlantic MOC index by averaging the meridional overturning in a latitude band between 1km and 2km or any other index averaging the meridional overturning in a given basin and a given domain. See *AreaMoc*

Warning: The MOC for the given basin must be calculated previously. Usually, it will suffice to call the 'moc' diagnostic earlier in the DIAGS list.

Options:

1. **Min latitude:** Minimum latitude to compute
2. **Max latitude:** Maximum latitude to compute
3. **Min depth:** Minimum depth (in levels)
4. **Max depth:** Maximum depth (in levels)
5. **Basin = 'Global':** Basin to calculate the diagnostic on.

3.2.2 averagesection

Compute an average of a given zone. The variable MUST be in a regular grid See *AverageSection*

Options:

1. **Variable:** Variable to average
2. **Min longitude:** Minimum longitude to compute
3. **Max longitude:** Maximum longitude to compute
4. **Min latitude:** Minimum latitude to compute
5. **Max latitude:** Maximum latitude to compute
6. **Domain = ocean:** Variable domain

3.2.3 convectionsites

Compute the intensity of convection in the four main convection sites. See *ConvectionSites*

Options:

This diagnostic has no options

3.2.4 cutsection

Cuts a meridional or zonal section. See *CutSection*

Options:

1. **Variable:** Variable to cut the section on
2. **Zonal:** If True, calculates a zonal section. If False, it will be a meridional one
3. **Value:** Reference value for the section

4. **Domain = ocean:** Variable's domain

3.2.5 gyres

Compute the intensity of the subtropical and subpolar gyres. See *Gyres*

Options:

This diagnostic has no options

3.2.6 heatcontent

Compute the total and mean ocean heat content. See *HeatContent*

Options:

1. **Basin** Basin to calculate the heat content one
2. **Mixed layer:** If 1, reduces the compuation to the mixed layer. If -1, excludes the mixed layer from the computations. If 0, no effect.
3. **Min depth:** Minimum depth for the calculation in levels. If 0, whole depth is used
4. **Max depth:** Maximum depth for the calculation in levels

3.2.7 heatcontentlayer

Point-wise Ocean Heat Content in a specified ocean thickness. See *HeatContentLayer*

Options:

3. **Min depth:** Minimum depth for the calculation in meteres
4. **Max depth:** Maximum depth for the calculation in meters
5. **Basin = 'Global':** Basin to calculate the heat content on.

3.2.8 interpolate

3-dimensional conservative interpolation to the regular atmospheric grid. It can also be used for 2D (i,j) variables. See *Interpolate*

Warning: This interpolation requires the pre-generated weights that can be found in '/esnas/autosubmit/con_files/weights'. Make sure that they are available for your configuration.

Options:

1. **Target grid:** New grid for the data
2. **Variable:** Variable to interpolate
3. **Domain = ocean:** Variable's domain
4. **Invert latitude:** If True, inverts the latitude in the output file.
5. **Original grid = '':** Source grid to choose. By default this is the original data, but sometimes you will want to use another (for example, the 'rotated' one produced by the rotation diagnostic)

3.2.9 interpolateCDO

Bilinear interpolation to a given grid using CDO. See *InterpolateCDO*

Warning: This interpolation is non-conservative, so treat its output with care. It has the advantage that does not require the pre-generated weights so it can be used when the 'interp' diagnostic is not available.

Options:

1. **Variable:** variable to interpolate
2. **Target grid:** Variable domain
3. **Domain = ocean:** Variable's domain
4. **Mask oceans = True:** If True, replaces the values in the ocean by NaN. You must only set it to false if, for some reason, you are interpolating an atmospheric or land variable that is stored in the NEMO grid (yes, this can happen, i.e. with tas).
5. **Original grid = '':** Source grid to choose. By default this is the original data, but sometimes you will want to use another (for example, the 'rotated' one produced by the rotation diagnostic)

3.2.10 maskland

Replaces all values excluded by the mask by NaN. See `MaskLand`

Options:

1. **Domain:** Variable to mask domain
2. **Variable:** variable to mask
3. **Cell point = T:** Cell point where variable is stored. Options: T, U, V, W, F
4. **Original grid = '':** Source grid to choose. By default this is the original data, but sometimes you will want to use another (for example, the 'rotated' one produced by the rotation diagnostic)

3.2.11 maxmoc

Compute an Atlantic MOC index by finding the maximum of the annual mean meridional overturning in a latitude / depth region. Output from this diagnostic will be always in yearly frequency. See *MaxMoc*

Warning: The MOC for the given basin must be calculated previously. Usually, it will suffice to call the 'moc' diagnostic earlier in the DIAGS list.

Warning: This diagnostic can only be computed for full years. It will discard incomplete years and only compute the index in those with the full 12 months available.

Options:

1. **Min latitude:** Minimum latitude to compute
2. **Max latitude:** Maximum latitude to compute
3. **Min depth:** Minimum depth (in levels)
4. **Max depth:** Maximum depth (in levels)
5. **Basin = 'Global':** Basin to calculate the diagnostic on.

3.2.12 mixedlayerheatcontent

Compute mixed layer heat content. See *MixedLayerHeatContent*

Options:

This diagnostic has no options

3.2.13 mixedlayersaltcontent

Compute mixed layer salt content. See *MixedLayerSaltContent*

Options:

This diagnostic has no options

3.2.14 moc

Compute the MOC for oceanic basins. Required for 'areamoc' and 'maxmoc' See *Moc*

Options:

This diagnostic has no options

3.2.15 mxl

Compute the mixed layer depth. See *Mxl*

Options:

This diagnostic has no options

3.2.16 psi

Compute the barotropic stream function. See *Psi*

Options:

This diagnostic has no options

3.2.17 regmean

Computes the mean value of the field (3D, weighted). For 3D fields, a horizontal mean for each level is also given. If a spatial window is specified, the mean value is computed only in this window. See `RegionMean`

Options:

1. **Domain:** Variable domain
2. **Variable:** Variable to average
3. **Grid_point:** NEMO grid point used to store the variable: T, U, V ...
4. **Basin = Global:** Basin to compute
5. **Save 3d = True:** If True, it also stores the average per level
6. **Min depth:** Minimum depth to compute in levels. If -1, average from the surface
7. **Max depth:** Maximum depth to compute in levels. If -1, average to the bottom
8. **Variance = False:** If True, it also stores the variance
9. **Original grid = '':** Source grid to choose. By default this is the original data, but sometimes you will want to use another (for example, the 'rotated' one produced by the rotation diagnostic)

3.2.18 rotate

Rotates the given variables See *Rotation*

Options:

1. **Variable u:** Variable's u component
2. **Variable v:** Variable's u component
3. **Domain = ocean:** Variable domain:

4. **Executable = /home/Earth/jvegas/pyCharm/cfutools/interpolation/rotateUVorca:** Path to the executable that will compute the rotation

Warning: This default executable has been compiled for ORCA1 experiments. For other resolutions you must use other executables compiled ad-hoc for them

3.2.19 siasiesiv

Compute the sea ice extent , area and volume in both hemispheres or a specified region. See *Siasiesiv*

Options:

1. **Basin = 'Global':** Basin to restrict the computation to.

3.2.20 vgrad

Calculates the gradient between two levels in a 3D ocean variable. See *VerticalGradient*

Options:

1. **Variable:** Variable to compute
2. **Upper level = 1:** Upper level. Will be used as the reference to compute the gradient
3. **Lower level = 2:** Lower level.

3.2.21 verticalmean

Chooses vertical level in ocean, or vertically averages between 2 or more ocean levels. See *VerticalMean*

Options:

1. **Variable:** Variable to average
2. **Min depth = -1:** Minimum level to compute. If -1, average from the surface
3. **Max depth:** Maximum level to compute. If -1, average to the bottom

3.2.22 verticalmeanmeters

Averages vertically any given variable. See *VerticalMeanMeters*

Options:

1. **Variable:** Variable to average
2. **Min depth = -1:** Minimum depth to compute in meters. If -1, average from the surface
3. **Max depth:** Maximum depth to compute in meters. If -1, average to the bottom

3.3 Statistics

3.3.1 climpercent

Calculates the specified climatological percentile of a given variable. See *ClimatologicalPercentile*

Options:

1. **Domain:** Variable's domain
2. **Variable:** Variable to compute diagnostic on
3. **Leadtimes:** Leadtimes to compute
4. **Bins:** Number of bins to use to discretize the variable

3.3.2 monpercent

Calculates the specified monthly percentile of a given variable. See *MonthlyPercentile*

Options:

1. **Domain:** Variable's domain
2. **Variable:** Variable to compute diagnostic on
3. **Percentiles:** List of requested percentiles ('-' separated)

CHAPTER 4

Tips and tricks

4.1 Working with ORCA1

If you plan to run diagnostics for ORCA1 resolution, be aware that your workstation will be more than capable to run them. At this resolution, memory and CPU consumption is low enough to allow you keep using the machine while running, specially if you reserve a pair of cores for other uses.

4.2 Configuring core usage

By default, the Earth Diagnostics creates a thread for each available core for the execution. If you are using a queueing system, the diagnostics will always use the number of cores that you reserved. If you are running outside a queueing system, the diagnostics will try to use all the cores on the machine. To avoid this, add the MAX_CORES parameter to the DIAGNOSTICS section inside the diags.conf file that you are using.

4.3 Cleaning temp file

By default, EarthDiagnostics removes the temporary directory after execution. This behaviour can be avoided be setting ra

By default

```
earthdiags -f PATH_TO_CONF --clean
```

CHAPTER 5

What to do if you have an error

Sometimes, the diagnostics may crash and you will not know why. This section will give you a procedure to follow before reporting the issue. This procedure is intended to solve some common problems or, at least, to help you in creating good issue reports. Remember: a good issue report reduces the time required to solve it!

Hint: Please, read carefully the error message. Most times the error message will point you to the problem's source and sometimes even give you a hint of how to solve it by yourself. And if this it not the case or if you find it obscure, even if it was helpful, please contact the developers so it can be improved in further versions

Try this simple steps BEFORE reporting an issue

- Clean scratch folder
- Update to the latest compatible tag: maybe your issue is already solved in it
- If you get the error for the first chunk of a given diagnostic, change the number of chunks to 1
- Call the diags with the -lc DEBUG -log log.txt options

Now, you have two options: if everything is fine, the error was probably due to some corrupted files or some unstable machine state. Nevertheless, try running the diagnostic with -lc DEBUG -log log.txt for all the chunks. If everything it's fine that's all.

If you experienced the same problem again, go to the GitLab portal and look into the open issues (https://earth.bsc.es/gitlab/es/ocean_diagnostics/issues). If you find your issue or a very similar one, use it to report your problems. If you can not find an open one that suites your problem, create a new one and explain what is happening to you. In any case, it will be very useful if you can attach your diags.conf and log.txt files and specify the machine you were using.

After that, it's just a matter of waiting for the developers to do their work and answering the questions that they may have. Please, be patient.

Caution: Of course, there is a third option: you keep experiencing an error that appears randomly on some executions but you are not able to reproduce it in a consistent manner. Report it and attach as much logs and configuration files as you have, along with the date and time of the errors.

CHAPTER 6

Developer's guide

The tool provides a set of useful diagnostics, but a lot more can be required at anytime. If you miss something and are able to develop it, you are more than welcome to collaborate. Even if you can not develop it, please let us know what do you want.

The first step is to go to the GitLab page for the project (https://earth.bsc.es/gitlab/es/ocean_diagnostics/) and open a new issue. Be sure that the title is self-explicative and give a detailed description of what you want. Please, be very explicit about what you want to avoid misunderstandings.

Hint: If reading your description, you think that you are taking the developers as stupids, you are doing it perfectly.

Don't forget to add the relevant tags. At this stage you will have to choose between 'enhancement', if you are proposing an improvement on a currently available feature, or 'new feature' in any the other case.

Now, if you are thinking on developing it yourself, please refer to the BSC-ES Git strategy (wiki_link_when_available) If you have any doubts, or just want help to start the development, contact javier.vegas@bsc.es.

6.1 Developing a diagnostic

For new diagnostics development, we have some advice to give:

- Do not worry about performance at first, just create a version that works. Developers can help you to optimize it later.
- There is nothing wrong with doing some common preparations in the generate_jobs of the diagnostic.
- Parallelization is achieved by running multiple diagnostics at a time. You don't need to implement it at diagnostic level
- Use the smallest time frame for your diagnostic: if you can work at chunk level, do not ask for full year data.
- Prefer NCO over CDO, you will have less problems when versions change.
- Ask for help as soon as you get stuck.

- Use always the methods in Utils instead of writing your own code.
- Use meaningful variable names. If you are using short names just to write less, please switch to an editor with autocompletion!
- Do not modify the mesh and mask files, another diagnostic can be using them at the same time.

CHAPTER 7

Frequently Asked Questions

Here will be the answers to the most usual questions. For the moment, there is nothing to see here…

CHAPTER 8

Module documentation

8.1 earthdiagnostics

8.1.1 earthdiagnostics.box

Module to manage 3D space restrictions

class earthdiagnostics.box.**Box**(*depth_in_meters=False*)

Bases: object

Represents a box in the 3D space.

Also allows easy conversion from the coordinate values to significant string representations

Parameters **depth_in_meters** (*bool, optional*) – If True, depth is given in meters. If False, it correspond to levels

depth_in_meters = None

If True, treats the depth as if it is given in meters. If False, as it is given in levels :rtype: bool

get_depth_str()

Get a string representation of depth.

For depth expressed in meters, it adds the character 'm' to the end If min_depth is different from max_depth, it concatenates the two values

Returns string representation for depth

Return type str

get_lat_str()

Get a string representation of the latitude in the format XX{N/S}.

If min_lat is different from max_lat, it concatenates the two values

Returns string representation for latitude

Return type str

get_lon_str()
Get a string representation of the longitude in the format XX{E/W}.

If min_lon is different from max_lon, it concatenates the two values

Returns string representation for longitude

Return type str

max_depth = None
Maximum depth :rtype: float

max_lat
Maximum latitude

Return type float

max_lon
Maximum longitude

Return type float

min_depth = None
Minimum depth :rtype: float

min_lat
Minimum latitude

Return type float

min_lon
Minimum longitude

Return type float

8.1.2 earthdiagnostics.cdftools

CDFTOOLS interface

class earthdiagnostics.cdftools.**CDFTools**(*path=''*)
Bases: object

Class to run CDFTools executables

Parameters **path** (*str*) – path to CDFTOOLS binaries

run(*command*, *input_file*, *output_file=None*, *options=None*, *log_level=20*, *input_option=None*)
Run one of the CDFTools

Parameters

- **command** (*str | iterable*) – executable to run
- **input_file** (*str*) – input file
- **output_file** – output file. Not all tools support this parameter
- **options** (*str | [str] | Tuple[str] | None*) – options for the tool.
- **log_level** (*int*) – log level at which the output of the cdftool command will be added
- **input_option** (*str*) – option to add before input file

8.1.3 earthdiagnostics.cmorizer

8.1.4 earthdiagnostics.cmormanager

8.1.5 earthdiagnostics.config

Classes to manage Earth Diagnostics configuration

class earthdiagnostics.config.**CMORConfig**(*parser*, *var_manager*)
Bases: object

Configuration for the cmorization processes

Parameters

- **parser** (*ConfigParser*) –
- **var_manager** (VariableManager) –

any_required(*variables*)
Check if any of the given variables is needed for cmorization

Parameters **variables** (*iterable of str*) –

Returns

Return type bool

chunk_cmorization_requested(*chunk*)
Check if the cmorization of a given chunk is required

Parameters **chunk** (*int*) –

Returns

Return type bool

cmorize(*var_cmor*)
Check if var_cmor is on variable list

Parameters **var_cmor** (Variable) –

get_levels(*frequency*, *variable*)
Get the levels to extract for a given variable

Parameters

- **frequency** (Frequency) –
- **variable** (*str*) –

Returns

Return type iterable of int

get_requested_codes()
Get all the codes to be extracted from the grib files

Returns

Return type set of int

get_variables(*frequency*)
Get the variables to get from the grib file for a given frequency

Parameters **frequency** (Frequency) –

Returns

Return type str

Raises ValueError – If the frequency passed is not supported

class earthdiagnostics.config.**Config**

Bases: object

Class to read and manage the configuration

auto_clean = None
If True, the scratch dir is removed after finishing

cdftools_path = None
Path to CDFTOOLS executables

cmor = None
CMOR related configuration

Returns

Return type *CMORConfig*

con_files = None
Mask and meshes folder path

data_adaptor = None
Scratch folder path

data_convention = None
Data convention to use

data_dir = None
Root data folder path

data_type = None
Data type (experiment, observation or reconstruction)

experiment = None
Configuration related to the experiment

Returns

Return type *ExperimentConfig*

frequency = None
Default data frequency to be used by the diagnostics

get_commands()
Return the list of commands after replacing the alias

Returns

Return type iterable of str

mask_regions = None
Custom mask regions file to use

mask_regions_3d = None
Custom mask regions 3D file to use

max_cores = None
Maximum number of cores to use

mesh_mask = None
Custom mesh mask file to use

new_mask_glo = None
Custom new mask glo file to use

parallel_downloads = None
Maximum number of simultaneous downloads

parallel_uploads = None
Maximum number of simultaneous uploads

parse(*path*)
Read configuration from INI file

Parameters **path** (*str*) –

report = None
Reporting configuration

Returns

Return type *ReportConfig*

restore_meshes = None
If True, forces the tool to copy all the mesh and mask files for the model, regardless of existence

scratch_dir = None
Scratch folder path

scratch_masks = None
Common scratch folder for masks

skip_diags_done = None
Flag to control if already done diags must be recalculated

thredds = None
THREDDS server configuration

Returns

Return type *THREDDSConfig*

use_ramdisk = None
If True, the scratch dir is created as a ram disk

exception earthdiagnostics.config.**ConfigException**
Bases: exceptions.Exception

Exception raised when there is a problem with the configuration

class earthdiagnostics.config.**ExperimentConfig**
Bases: object

Configuration related to the experiment

get_chunk_end(*startdate*, *chunk*)
Get chunk's last day

Parameters

- **startdate** (*str or datetime.datetime*) –
- **chunk** (*int*) –

Returns

Return type datetime.datetime

See also:

get_chunk_end_str()

get_chunk_end_str(*startdate*, *chunk*)
Get chunk's last day as a string

Parameters

- **startdate** (*str or datetime.datetime*) –
- **chunk** (*int*) –

Returns

Return type datetime.datetime

See also:

get_chunk_end()

get_chunk_list()
Return a list with all the chunks

Returns List containing tuples of startdate, member and chunk

Return type tuple[str, int, int]

get_chunk_start(*startdate*, *chunk*)
Get chunk's first day

Parameters

- **startdate** (*str or datetime.datetime*) –
- **chunk** (*int*) –

Returns

Return type datetime.datetime

See also:

get_chunk_start_str()

get_chunk_start_str(*startdate*, *chunk*)
Get chunk's first day string representation

Parameters

- **startdate** (*str or datetime.datetime*) –
- **chunk** (*int*) –

Returns

Return type str

See also:

get_chunk_start()

get_full_years(*startdate*)
Return the list of full years that are in the given startdate

Parameters **startdate** (*str*) – startdate to use

Returns list of full years

Return type list[int]

get_member_list()
Return a list with all the members

Returns List containing tuples of startdate and member

Return type tuple[str, int, int]

get_member_str(*member*)
Return the member name for a given member number.

Parameters **member** (*int*) – member's number

Returns member's name

Return type str

get_year_chunks(*startdate*, *year*)
Get the list of chunks containing timesteps from the given year

Parameters

- **startdate** (*str*) – startdate to use
- **year** (*int*) – reference year

Returns list of chunks containing data from the given year

Return type list[int]

parse_ini(*parser*)
Parse experiment section from INI-like file

Parameters **parser** (*ConfigParser*) –

class earthdiagnostics.config.**ReportConfig**(*parser*)
Bases: object

Configuration for the reporting feature

Parameters **parser** (*ConfigParser*) –

class earthdiagnostics.config.**THREDDSConfig**(*parser*)
Bases: object

Configuration related to the THREDDS server

Parameters **parser** (*ConfigParser*) –

8.1.6 earthdiagnostics.constants

Contains the enumeration-like classes used by the diagnostics

class earthdiagnostics.constants.**Basin**(*name*)
Bases: object

Class representing a given basin

Parameters **name** (*str*) – full basin's name

name
Basin full name

Return type str

class earthdiagnostics.constants.**Basins**
Bases: object

Singleton class to manage available basins

get_available_basins(*handler*)
Read available basins from file

Parameters **handler**(*netCDF4.Dataset*) –

parse(*basin*)
Return the basin matching the given name.

If the parameter basin is a Basin instance, directly returns the same instance. This bahaviour is intended to facilitate the development of methods that can either accept a nameor a Basin instance to characterize the basin.

Parameters **basin**(*str | Basin*) – basin name or basin instance

Returns basin instance corresponding to the basin name

Return type *Basin*

class earthdiagnostics.constants.**Models**
Bases: object

Predefined models

ECEARTH_2_3_O1L42 = 'Ec2.3_O1L42'
EC-Earth 2.3 ORCA1 L42

ECEARTH_3_0_O1L46 = 'Ec3.0_O1L46'
EC-Earth 3 ORCA1 L46

ECEARTH_3_0_O25L46 = 'Ec3.0_O25L46'
EC-Earth 3 ORCA0.25 L46

ECEARTH_3_0_O25L75 = 'Ec3.0_O25L75'
EC-Earth 3 ORCA0.25 L75

ECEARTH_3_1_O25L75 = 'Ec3.1_O25L75'
EC-Earth 3.1 ORCA0.25 L75

ECEARTH_3_2_O1L75 = 'Ec3.2_O1L75'
EC-Earth 3.2 ORCA1 L75

ECEARTH_3_2_O25L75 = 'Ec3.2_O25L75'
EC-Earth 3.2 ORCA0.25 L75

GLORYS2_V1_O25L75 = 'glorys2v1_O25L75'
GLORYS2v1 ORCA0.25 L75

NEMOVAR_O1L42 = 'nemovar_O1L42'
NEMOVAR ORCA1 L42

NEMO_3_2_O1L42 = 'N3.2_O1L42'
NEMO 3.2 ORCA1 L42

NEMO_3_3_O1L46 = 'N3.3_O1L46'
NEMO 3.3 ORCA1 L46

NEMO_3_6_O1L46 = 'N3.6_O1L75'
NEMO 3.6 ORCA1 L75

8.1.7 earthdiagnostics.datafile

Module for classes to manage storage manipulation

class earthdiagnostics.datafile.**DataFile**
Bases: *earthdiagnostics.publisher.Publisher*

Represent a data file

Must be derived for each concrete data file format

add_cmorization_history()
Add the history line corresponding to the cmorization to the local file

add_diagnostic_history()
Add the history line corresponding to the diagnostic to the local file

add_modifier(*diagnostic*)
Register a diagnostic as a modifier of this data

A modifier diagnostic is a diagnostic that read this data and changes it in any way. The diagnostic must be a modifier even if it only affects the metadata

Parameters **diagnostic** (Diagnostic) –

clean_local()
Check if a local file is still needed and remove it if not

create_link()
Create a link from the original data in the <frequency>_<var_type> folder

dispatch(**args*)
Notify update to all the suscribers

Parameters **args** – arguments to pass

download()
Get data from remote storage to the local one

Must be overriden by the derived classes

Raises NotImplementedError – If the derived classes do not override this

download_required()
Get if a download is required for this file

Returns

Return type bool

classmethod from_storage(*filepath*, *data_convention*)
Create a new datafile to be downloaded from the storage

has_modifiers()
Check if it has registered modifiers

Returns

Return type bool

local_status
Get local storage status

only_suscriber(*who*)
Get if an object is the sole suscriber of this publisher

Parameters **who** (*object*) –

Returns

Return type bool

prepare_to_upload(*rename_var*)
Prepare a local file to be uploaded

This includes renaming the variable if necessary, updating the metadata and adding the history and managing the possibility of multiple regions

ready_to_run(*diagnostic*)
Check if the data is ready to run for a given diagnostics

To be ready to run, the datafile should be in the local storage and no modifiers can be pending.

Parameters **diagnostic** (Diagnostic) –

Returns

Return type bool

set_local_file(*local_file*, *diagnostic=None*, *rename_var=''*, *region=None*)
Set the local file generated by EarthDiagnostics

This also prepares it for the upload

Parameters

- **local_file** (*str*) –
- **diagnostic** (Diagnostic *or None*) –
- **rename_var** (*str*) –
- **region** (Basin *or None*) –

Returns

Return type None

size
File size

storage_status
Get remote storage status

subscribe(*who*, *callback=None*)
Add a suscriber to the current publisher

Parameters

- **who** (*object*) – Subscriber to add
- **callback** (*callable or None, optional*) – Callback to call

suscribers
List of suscribers of this publisher

classmethod to_storage(*remote_file*, *data_convention*)
Create a new datafile object for a file that is going to be generated and stored

unsubscribe(*who*)
Remove a suscriber from the current publisher

Parameters **who** (*object*) – suscriber to remove

upload()
Send a loal file to the storage

upload_required()
Get if an upload is needed for this file

Returns

Return type bool

class earthdiagnostics.datafile.**LocalStatus**
Bases: object

Local file status enumeration

class earthdiagnostics.datafile.**NetCDFFile**
Bases: *earthdiagnostics.datafile.DataFile*

Implementation of DataFile for netCDF files

add_cmorization_history()
Add the history line corresponding to the cmorization to the local file

add_diagnostic_history()
Add the history line corresponding to the diagnostic to the local file

add_modifier(*diagnostic*)
Register a diagnostic as a modifier of this data

A modifier diagnostic is a diagnostic that read this data and changes it in any way. The diagnostic must be a modifier even if it only affects the metadata

Parameters **diagnostic** (Diagnostic) –

clean_local()
Check if a local file is still needed and remove it if not

create_link()
Create a link from the original data in the <frequency>_<var_type> folder

dispatch(**args*)
Notify update to all the suscribers

Parameters **args** – arguments to pass

download()
Get data from remote storage to the local one

download_required()
Get if a download is required for this file

Returns

Return type bool

classmethod from_storage(*filepath*, *data_convention*)
Create a new datafile to be downloaded from the storage

has_modifiers()
Check if it has registered modifiers

Returns

Return type bool

local_status
Get local storage status

only_suscriber(*who*)
Get if an object is the sole suscriber of this publisher

Parameters **who** (*object*) –

Returns

Return type bool

prepare_to_upload(*rename_var*)
Prepare a local file to be uploaded

This includes renaming the variable if necessary, updating the metadata and adding the history and managing the possibility of multiple regions

ready_to_run(*diagnostic*)
Check if the data is ready to run for a given diagnostics

To be ready to run, the datafile should be in the local storage and no modifiers can be pending.

Parameters **diagnostic** (Diagnostic) –

Returns

Return type bool

set_local_file(*local_file*, *diagnostic=None*, *rename_var=''*, *region=None*)
Set the local file generated by EarthDiagnostics

This also prepares it for the upload

Parameters

- **local_file** (*str*) –
- **diagnostic** (Diagnostic *or None*) –
- **rename_var** (*str*) –
- **region** (Basin *or None*) –

Returns

Return type None

size
File size

storage_status
Get remote storage status

subscribe(*who*, *callback=None*)
Add a suscriber to the current publisher

Parameters

- **who** (*object*) – Subscriber to add
- **callback** (*callable or None, optional*) – Callback to call

suscribers
List of suscribers of this publisher

classmethod to_storage(*remote_file*, *data_convention*)
Create a new datafile object for a file that is going to be generated and stored

unsubscribe(*who*)
Remove a suscriber from the current publisher

Parameters **who** (*object*) – suscriber to remove

upload()
Send a loal file to the storage

upload_required()
Get if an upload is needed for this file

Returns

Return type bool

class earthdiagnostics.datafile.**StorageStatus**
Bases: object

Remote file status enumeration

class earthdiagnostics.datafile.**UnitConversion**(*source*, *destiny*, *factor*, *offset*)
Bases: object

Class to manage unit conversions

Parameters

- **source** (*str*) –
- **destiny** (*str*) –
- **factor** (*float*) –
- **offset** (*float*) –

classmethod add_conversion(*conversion*)
Add a conversion to the dictionary

Parameters **conversion** (UnitConversion) – conversion to add

classmethod get_conversion_factor_offset(*input_units*, *output_units*)
Get the conversion factor and offset for two units.

The conversion has to be done in the following way: converted = original * factor + offset

Parameters

- **input_units** (*str*) – original units
- **output_units** (*str*) – destiny units

Returns factor and offset

Return type [float, float]

classmethod load_conversions()
Load conversions from the configuration file

8.1.8 earthdiagnostics.datamanager

Base data manager for Earth diagnostics

class earthdiagnostics.datamanager.**DataManager**(*config*)
Bases: object

Class to manage the data repositories

Parameters **config** (Config) –

declare_chunk(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *region=None*, *box=None*, *frequency=None*, *vartype=1*, *diagnostic=None*)
Declare a variable chunk to be generated by a diagnostic

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **chunk** (*int*) –
- **grid** (*str or None, optional*) –
- **region** (Basin *or None, optional*) –
- **box** (Box *or None, optional*) –
- **frequency** (Frequency *or None, optional*) –
- **vartype** (VariableType, *optional*) –
- **diagnostic** (Diagnostic, *optional*) –

Returns

Return type *DataFile*

Raises NotImplementedError – If not implemented by derived classes

declare_year(*domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*, *vartype=1*, *diagnostic=None*)
Declare a variable year to be generated by a diagnostic

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **year** (*int*) –
- **grid** (*str or None, optional*) –
- **box** (Box *or None, optional*) –
- **vartype** (VariableType, *optional*) –
- **diagnostic** (Diagnostic, *optional*) –

Returns

Return type *DataFile*

Raises NotImplementedError – If not implemented by derived classes

file_exists(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *box=None*, *frequency=None*, *vartype=1*, *possible_versions=None*)
Check if a file exists in the storage

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **chunk** (*int*) –
- **grid** (*str or None, optional*) –
- **box** (*Box or None, optional*) –
- **frequency** (*Frequency or None, optional*) –
- **vartype** (*VariableType, optional*) –
- **possible_versions** (*iterable od str or None, optional*) –

Raises NotImplementedError – If not implemented by derived classes

Returns

Return type bool

link_file(*domain*, *var*, *cmor_var*, *startdate*, *member*, *chunk=None*, *grid=None*, *frequency=None*, *year=None*, *date_str=None*, *move_old=False*, *vartype=1*)

Create the link of a given file from the CMOR repository.

Parameters

- **cmor_var** –
- **move_old** –
- **date_str** –
- **year** (*int*) – if frequency is yearly, this parameter is used to give the corresponding year
- **domain** (*Domain*) – CMOR domain
- **var** (*str*) – variable name
- **startdate** (*str*) – file's startdate
- **member** (*int*) – file's member
- **chunk** (*int*) – file's chunk
- **grid** (*str*) – file's grid (only needed if it is not the original)
- **frequency** (*str*) – file's frequency (only needed if it is different from the default)
- **vartype** (VariableType) – Variable type (mean, statistic)

Returns path to the copy created on the scratch folder

Return type str

prepare()

Prepare the data to be used by Earth Diagnostics

request_chunk(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *box=None*, *frequency=None*, *vartype=None*)

Request a given file from the CMOR repository to the scratch folder and returns the path to the scratch's copy

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **chunk** (*int*) –
- **grid** (*str or None, optional*) –
- **box** (*Box or None, optional*) –
- **frequency** (*Frequency or None, optional*) –
- **vartype** (*VariableType or None, optional*) –

Returns

Return type *DataFile*

Raises NotImplementedError – If not implemented by derived classes

request_year(*diagnostic*, *domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*, *frequency=None*)

Request a given year for a variavle from a CMOR repository

Parameters

- **diagnostic** (*Diagnostic*) –
- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **year** (*int*) –
- **grid** (*str or None, optional*) –
- **box** (*Box or None, optional*) –
- **frequency** (*Frequency or None, optional*) –

Returns

Return type *DataFile*

Raises NotImplementedError – If not implemented by derived classes

8.1.9 earthdiagnostics.diagnostic

This module contains the Diagnostic base class and all the classes for parsing the options passed to them

class earthdiagnostics.diagnostic.**Diagnostic**(*data_manager*)

Bases: *earthdiagnostics.publisher.Publisher*

Base class for the diagnostics.

Provides a common interface for them and also has a mechanism that allows diagnostic retrieval by name.

Parameters **data_manager** (DataManager) – data manager that will be used to store and retrieve the necessary data

add_subjob(*subjob*)
Add a subjob

Add a diagnostic that must be run before the current one

Parameters **subjob** (Diagnostic) –

alias = None
Alias to call the diagnostic. Must be overridden at the derived clases

all_requests_in_storage()
Check if all the data requested is in the local scratch

Returns

Return type bool

can_skip_run()
Check if a diagnostic calculation can be skipped

Looks if the data to be generated is already there and is not going to be modified

Returns

Return type bool

check_is_ready()
Check if a diagnostic is ready to run and change its status accordingly

compute()
Calculate the diagnostic and stores the output

Must be implemented by derived classes

declare_chunk(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *region=None*, *box=None*, *frequency=None*, *vartype=1*)
Declare a chunk that is going to be generated by the diagnostic

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int or None*) –
- **chunk** (*int or None*) –
- **grid** (*str or None*) –
- **region** (Basin *or None*) –
- **box** (Box *or None*) –
- **frequency** (Frequency *or None*) –
- **vartype** (VariableType) –

Returns

Return type *DataFile*

declare_data_generated()
Declare the data to be generated by the diagnostic

Must be implemented by derived classes

declare_year(*domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*, *vartype=1*)
Declare a year that is going to be generated by the diagnostic

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **year** (*int*) –
- **grid** (*str or None*) –
- **box** (*Box or None*) –
- **vartype** (*VariableType*) –

Returns

Return type *DataFile*

dispatch(**args*)
Notify update to all the suscribers

Parameters **args** – arguments to pass

classmethod generate_jobs(*diags*, *options*)
Generate the instances of the diagnostics that will be run by the manager

Must be implemented by derived classes.

Parameters

- **diags** (*Diags*) –
- **options** (*list of str*) –

Returns

Return type list of Diagnostic

static get_diagnostic(*name*)
Return the class for a diagnostic given its name

Parameters **name** (*str*) –

Returns

Return type Type[*Diagnostic*] or None

only_suscriber(*who*)
Get if an object is the sole suscriber of this publisher

Parameters **who** (*object*) –

Returns

Return type bool

pending_requests()
Get the number of data request pending to be fulfilled

Returns

Return type int

classmethod process_options(*options*, *options_available*)
Process the configuration of a diagnostic

Parameters

- **options** (*iterable of str*) –
- **options_available** (*iterable of DiagnosticOptiion*) –

Returns **dict of str** – Dictionary of names and values for the options

Return type str

Raises DiagnosticOptionError: – If there are more options that admitted for the diagnostic

static register(*diagnostic_class*)
Register a new diagnostic using the given alias.

It must be called using the derived class.

Parameters **diagnostic_class** (*Type[Diagnostic]*) –

request_chunk(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *box=None*, *frequency=None*, *to_modify=False*, *vartype=1*)
Request one chunk of data required by the diagnostic

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str or None*) –
- **member** (*int or None*) –
- **chunk** (*int or None*) –
- **grid** (*str or None*) –
- **box** (Box *or None*) –
- **frequency** (Frequency *or str or None*) –
- **to_modify** (*bool*) – Flag that must be active if the diagnostic is going to generate a modified version of this data. In this case this data must not be declared as an output of the diagnostic
- **vartype** (VariableType) –

Returns

Return type *DataFile*

See also:

request_year(), *declare_chunk()*, *declare_year()*

request_data()
Request the data required by the diagnostic

Must be implemented by derived classes

request_year(*domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*, *frequency=None*, *to_modify=False*)
Request one year of data that is required for the diagnostic

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **year** (*int*) –
- **grid** (*str*) –
- **box** (Box) –
- **frequency** (Frequency) –
- **to_modify** (*str*) –

Returns

Return type *DataFile*

See also:

request_chunk(), *declare_chunk()*, *declare_year()*

status
Execution status

subscribe(*who*, *callback=None*)
Add a suscriber to the current publisher

Parameters

- **who** (*object*) – Subscriber to add
- **callback** (*callable or None, optional*) – Callback to call

suscribers
List of suscribers of this publisher

unsubscribe(*who*)
Remove a suscriber from the current publisher

Parameters **who** (*object*) – suscriber to remove

class earthdiagnostics.diagnostic.**DiagnosticBasinListOption**(*name*, *default_value=None*)
Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class to parse list of basins options

parse(*option_value*)
Parse option value

Parameters **option_value** (*str*) –

Returns

Return type *Basin*

class earthdiagnostics.diagnostic.**DiagnosticBasinOption**(*name*, *default_value=None*)

Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class to parse basin options

parse(*option_value*)

Parse option value

Parameters **option_value** (*str*) –

Returns

Return type *Basin*

class earthdiagnostics.diagnostic.**DiagnosticBoolOption**(*name*, *default_value=None*)

Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class to parse boolean options

parse(*option_value*)

Parse option value

Parameters **option_value** (*str*) –

Returns

Return type Bool

class earthdiagnostics.diagnostic.**DiagnosticChoiceOption**(*name*, *choices*, *default_value=None*, *ignore_case=True*)

Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class to parse choice option

Parameters

- **name** (*str*) –
- **choices** (*list of str*) – Valid options for the option
- **default_value** (*str, optional*) – If not None, it should ve a valid choice
- **ignore_case** (*bool, optional*) – If false, value must match case of the valid choice

parse(*option_value*)

Parse option value

Parameters **option_value** (*str*) –

Returns

Return type str

class earthdiagnostics.diagnostic.**DiagnosticComplexStrOption**(*name*, *default_value=None*)

Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class to parse complex string options

It replaces '&;' with ',' and '&.' with ' '

parse(*option_value*)

Parse option value

Parameters **option_value** (*str*) –

Returns

Return type str

class earthdiagnostics.diagnostic.**DiagnosticDomainOption**(*name='domain'*, *default_value=None*)

Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class to parse domain options

Parameters

- **name** (*str, optional*) –
- **default_value** (*str, optional*) –

parse(*option_value*)

Parse option value

Returns

Return type ModelingRealm

class earthdiagnostics.diagnostic.**DiagnosticFloatOption**(*name*, *default_value=None*)

Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class for parsing float options

parse(*option_value*)

Parse option value

Parameters **option_value** (*str*) –

Returns

Return type float

class earthdiagnostics.diagnostic.**DiagnosticFrequencyOption**(*name='frequency'*, *default_value=None*)

Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class to parse frequency options

Parameters

- **name** (*str, optional*) –
- **default_value** (Frequency,*optional*) –

parse(*option_value*)

Parse option value

Parameters **option_value** (*str*) –

Returns

Return type *Frequency*

class earthdiagnostics.diagnostic.**DiagnosticIntOption**(*name*, *default_value=None*, *min_limit=None*, *max_limit=None*)

Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class for parsing integer options

Parameters

- **name** (*str*) –
- **default_value** (*int, optional*) –
- **min_limit** (*int, optional*) – If setted, any value below this will not be accepted
- **max_limit** (*int, optional*) – If setted, any value over this will not be accepted

parse(*option_value*)
Parse option value

Parameters **option_value** (*str*) –

Returns

Return type int

Raises *DiagnosticOptionError* – If parsed values is outside limits

class earthdiagnostics.diagnostic.**DiagnosticListFrequenciesOption**(*name*, *default_value=None*)
Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class for parsing an option which is a list of frequencies

Parameters

- **name** (*str*) –
- **default_value** (*list, optional*) –

parse(*option_value*)
Parse option value

Returns

Return type List of Frequency

class earthdiagnostics.diagnostic.**DiagnosticListIntOption**(*name*, *default_value=None*, *min_limit=None*, *max_limit=None*)
Bases: *earthdiagnostics.diagnostic.DiagnosticIntOption*

Class for parsing integer list options

Parameters

- **name** (*str*) –
- **default_value** (*list, optional*) –
- **min_limit** (*int, optional*) – If setted, any value below this will not be accepted
- **max_limit** (*int, optional*) – If setted, any value over this will not be accepted

max_limit = None
Upper limit

min_limit = None
Lower limit

parse(*option_value*)
Parse option value

Parameters **option_value** (*str*) –

Returns

Return type list(int)

Raises *DiagnosticOptionError* – If parsed values is outside limits

class earthdiagnostics.diagnostic.**DiagnosticOption**(*name*, *default_value=None*)
Bases: object

Class to manage string options for the diagnostic

parse(*option_value*)
Get the final value for the option

If option_value is empty, return default_value

Parameters **option_value** (*str*) –

Returns

Return type str

Raises DiagnosticOptionError: – If the option is empty and default_value is False

exception earthdiagnostics.diagnostic.**DiagnosticOptionError**
Bases: exceptions.Exception

Exception class for errors related to bad options for the diagnostics

class earthdiagnostics.diagnostic.**DiagnosticStatus**
Bases: object

Enumeration of diagnostic status

class earthdiagnostics.diagnostic.**DiagnosticVariableListOption**(*var_manager*, *name*, *default_value=None*)
Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class to parse variable list options

Parameters

- **var_manager** (VariableManager) –
- **name** (*str, optional*) –
- **default_value** (*str, optional*) –

parse(*option_value*)
Parse option value

Returns

Return type List[*Variable*]

class earthdiagnostics.diagnostic.**DiagnosticVariableOption**(*var_manager*, *name='variable'*, *default_value=None*)
Bases: *earthdiagnostics.diagnostic.DiagnosticOption*

Class to parse variable options

Parameters

- **var_manager** (VariableManager) –
- **name** (*str, optional*) –
- **default_value** (*str, optional*) –

parse(*option_value*)
Parse option value

Returns

Return type *Variable*

8.1.10 earthdiagnostics.earthdiags

8.1.11 earthdiagnostics.frequency

Data frequency management tools

class earthdiagnostics.frequency.**Frequencies**
Bases: object

Enumeration of supported frequencies

class earthdiagnostics.frequency.**Frequency**(*freq*)
Bases: object

Time frequency

folder_name(*vartype*)
Get foder name associated to this frequency

Parameters **vartype** (VariableType) –

Returns

Return type str

static parse(*freq*)
Get frequency instance from str

If a Frequency object is passed, it is returned as usual

Parameters **freq** (*str or* Frequency) –

Returns

Return type *Frequency*

8.1.12 earthdiagnostics.modellingrealm

8.1.13 earthdiagnostics.obsreconmanager

Data management for BSC-Earth conventions

Focused on working with observations and reconstructions as well as with downloaded but no cmorized models (like ECMWF System 4)

class earthdiagnostics.obsreconmanager.**ObsReconManager**(*config*)
Bases: *earthdiagnostics.datamanager.DataManager*

Data manager class for CMORized experiments

Parameters **config** (Config) –

declare_chunk(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *region=None*, *box=None*, *frequency=None*, *vartype=1*, *diagnostic=None*)
Declare a variable chunk to be generated by a diagnostic

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **chunk** (*int*) –
- **grid** (*str or None, optional*) –
- **region** (*Basin or None, optional*) –
- **box** (*Box or None, optional*) –
- **frequency** (*Frequency or None, optional*) –
- **vartype** (*VariableType, optional*) –
- **diagnostic** (*Diagnostic, optional*) –

Returns

Return type *DataFile*

declare_year(*domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*, *vartype=1*, *diagnostic=None*)

Declare a variable year to be generated by a diagnostic

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **year** (*int*) –
- **grid** (*str or None, optional*) –
- **box** (*Box or None, optional*) –
- **vartype** (*VariableType, optional*) –
- **diagnostic** (*Diagnostic, optional*) –

Returns

Return type *DataFile*

Raises NotImplementedError – If not implemented by derived classes

file_exists(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *box=None*, *frequency=None*, *vartype=1*, *possible_versions=None*)

Check if a file exists in the storage

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –

- **chunk** (*int*) –
- **grid** (*str or None, optional*) –
- **box** (*Box or None, optional*) –
- **frequency** (*Frequency or None, optional*) –
- **vartype** (*VariableType, optional*) –
- **possible_versions** (*iterable od str or None, optional*) –

Raises NotImplementedError – If not implemented by derived classes

Returns

Return type bool

get_file_path(*startdate*, *domain*, *var*, *frequency*, *vartype*, *box=None*, *grid=None*)
Return the path to a concrete file

Parameters

- **startdate** (*str*) – file's startdate
- **domain** (*str*) – file's domain
- **var** (*str*) – file's var
- **frequency** (*Frequency*) – file's frequency
- **box** (*Box*) – file's box
- **grid** (*str*) – file's grid
- **vartype** (*VariableType*) – Variable type (mean, statistic)

Returns path to the file

Return type str

link_file(*domain*, *var*, *cmor_var*, *startdate*, *member*, *chunk=None*, *grid=None*, *frequency=None*, *year=None*, *date_str=None*, *move_old=False*, *vartype=1*)
Create the link of a given file from the CMOR repository.

Parameters

- **cmor_var** –
- **move_old** –
- **date_str** –
- **year** (*int*) – if frequency is yearly, this parameter is used to give the corresponding year
- **domain** (*Domain*) – CMOR domain
- **var** (*str*) – variable name
- **startdate** (*str*) – file's startdate
- **member** (*int*) – file's member
- **chunk** (*int*) – file's chunk
- **grid** (*str*) – file's grid (only needed if it is not the original)
- **frequency** (*str*) – file's frequency (only needed if it is different from the default)
- **vartype** (*VariableType*) – Variable type (mean, statistic)

Returns path to the copy created on the scratch folder

Return type str

prepare()
Prepare the data to be used by Earth Diagnostics

request_chunk(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *box=None*, *frequency=None*, *vartype=1*)
Request a given file from the CMOR repository to the scratch folder and returns the path to the scratch's copy

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **chunk** (*int*) –
- **grid** (*str or None*) –
- **box** (*Box or None*) –
- **frequency** (*Frequency or None*) –
- **vartype** (*VariableType or None*) –

Returns

Return type *DataFile*

request_year(*diagnostic*, *domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*, *frequency=None*)
Request a given year for a variavle from a CMOR repository

Parameters

- **diagnostic** (*Diagnostic*) –
- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **year** (*int*) –
- **grid** (*str or None, optional*) –
- **box** (*Box or None, optional*) –
- **frequency** (*Frequency or None, optional*) –

Returns

Return type *DataFile*

Raises NotImplementedError – If not implemented by derived classes

8.1.14 earthdiagnostics.publisher

Module to allow classes to communicate when an event is produced

class earthdiagnostics.publisher.**Publisher**
Bases: object

Base class to provide functionality to notify updates to other objects

dispatch(**args*)
Notify update to all the suscribers

Parameters **args** – arguments to pass

only_suscriber(*who*)
Get if an object is the sole suscriber of this publisher

Parameters **who** (*object*) –

Returns

Return type bool

subscribe(*who*, *callback=None*)
Add a suscriber to the current publisher

Parameters

- **who** (*object*) – Subscriber to add
- **callback** (*callable or None, optional*) – Callback to call

suscribers
List of suscribers of this publisher

unsubscribe(*who*)
Remove a suscriber from the current publisher

Parameters **who** (*object*) – suscriber to remove

8.1.15 earthdiagnostics.singleton

8.1.16 earthdiagnostics.threddsmanager

Data manager for THREDDS server

exception earthdiagnostics.threddsmanager.**THREDDSError**
Bases: exceptions.Exception

Exception to be launched when a THREDDS related error is encounteredd

class earthdiagnostics.threddsmanager.**THREDDSManager**(*config*)
Bases: *earthdiagnostics.datamanager.DataManager*

Data manager class for THREDDS

Parameters **config** (Config) –

declare_chunk(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *region=None*, *box=None*, *frequency=None*, *vartype=1*, *diagnostic=None*)
Copy a given file from the CMOR repository to the scratch folder and returns the path to the scratch's copy

Parameters

- **diagnostic** –
- **region** –
- **domain** (*Domain*) – CMOR domain
- **var** (*str*) – variable name
- **startdate** (*str*) – file's startdate
- **member** (*int*) – file's member
- **chunk** (*int*) – file's chunk
- **grid** (*str|None*) – file's grid (only needed if it is not the original)
- **box** (Box) – file's box (only needed to retrieve sections or averages)
- **frequency** (*Frequency|None*) – file's frequency (only needed if it is different from the default)
- **vartype** (VariableType) – Variable type (mean, statistic)

Returns path to the copy created on the scratch folder

Return type str

declare_year(*domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*, *vartype=1*, *diagnostic=None*)

Declare a variable year to be generated by a diagnostic

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **year** (*int*) –
- **grid** (*str or None, optional*) –
- **box** (Box *or None, optional*) –
- **vartype** (VariableType, *optional*) –
- **diagnostic** (Diagnostic, *optional*) –

Returns

Return type *DataFile*

Raises NotImplementedError – If not implemented by derived classes

file_exists(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *box=None*, *frequency=None*, *vartype=1*, *possible_versions=None*)

Check if a file exists in the storage

Creates a THREDDSSubset and checks if it is accesible

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –

- **member** (*int*) –
- **chunk** (*int*) –
- **grid** (*str or None*) –
- **box** (*Box or None*) –
- **frequency** (*Frequency or None*) –
- **vartype** (*VariableType*) –

Returns

Return type *THREDDSSubset*

get_file_path(*startdate*, *domain*, *var*, *frequency*, *vartype*, *box=None*, *grid=None*)
Return the path to a concrete file

Parameters

- **startdate** (*str*) –
- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **frequency** (*Frequency*) –
- **vartype** (*VariableType*) –
- **box** (*Box or None, optional*) –
- **grid** (*str or None, optional*) –

Returns

Return type str

get_var_url(*var*, *startdate*, *frequency*, *box*, *vartype*)
Get url for dataset

Parameters

- **var** (*str*) – variable to retrieve
- **startdate** (*str*) – startdate to retrieve
- **frequency** (*Frequency | None*) – frequency to get:
- **box** (*Box*) – box to get
- **vartype** (*VariableType*) – type of variable

Returns

get_year(*domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*, *vartype=1*)
Ge a file containing all the data for one year for one variable

Parameters

- **domain** (*str*) – variable's domain
- **var** (*str*) – variable's name
- **startdate** (*str*) – startdate to retrieve
- **member** (*int*) – member to retrieve
- **year** (*int*) – year to retrieve

- **grid** (*str*) – variable's grid
- **box** (Box) – variable's box
- **vartype** (VariableType) – Variable type (mean, statistic)

Returns

link_file(*domain*, *var*, *cmor_var*, *startdate*, *member*, *chunk=None*, *grid=None*, *frequency=None*, *year=None*, *date_str=None*, *move_old=False*, *vartype=1*)

Create the link of a given file from the CMOR repository.

Parameters

- **cmor_var** –
- **move_old** –
- **date_str** –
- **year** (*int*) – if frequency is yearly, this parameter is used to give the corresponding year
- **domain** (*Domain*) – CMOR domain
- **var** (*str*) – variable name
- **startdate** (*str*) – file's startdate
- **member** (*int*) – file's member
- **chunk** (*int*) – file's chunk
- **grid** (*str*) – file's grid (only needed if it is not the original)
- **frequency** (*str*) – file's frequency (only needed if it is different from the default)
- **vartype** (VariableType) – Variable type (mean, statistic)

Returns path to the copy created on the scratch folder

Return type str

prepare()

Prepare the data to be used by Earth Diagnostics

request_chunk(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *box=None*, *frequency=None*, *vartype=1*)

Request a given file from the CMOR repository to the scratch folder and returns the path to the scratch's copy

Parameters

- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **chunk** (*int*) –
- **grid** (*str or None*) –
- **box** (*Box or None*) –
- **frequency** (*Frequency or None*) –
- **vartype** (*VariableType or None*) –

Returns

Return type *DataFile*

request_year(*diagnostic*, *domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*, *frequency=None*)

Request a given year for a variavle from a CMOR repository

Parameters

- **diagnostic** (Diagnostic) –
- **domain** (*ModelingRealm*) –
- **var** (*str*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **year** (*int*) –
- **grid** (*str or None, optional*) –
- **box** (Box *or None, optional*) –
- **frequency** (Frequency *or None, optional*) –

Returns

Return type *DataFile*

Raises NotImplementedError – If not implemented by derived classes

class earthdiagnostics.threddsmanager.**THREDDSSubset**(*thredds_path*, *file_path*, *var*, *start_time*, *end_time*)

Bases: *earthdiagnostics.datafile.DataFile*

Implementation of DataFile for the THREDDS server

Parameters

- **thredds_path** (*str*) –
- **file_path** (*str*) –
- **var** (*str*) –
- **start_time** (*datetime*) –
- **end_time** (*datetime*) –

add_cmorization_history()

Add the history line corresponding to the cmorization to the local file

add_diagnostic_history()

Add the history line corresponding to the diagnostic to the local file

add_modifier(*diagnostic*)

Register a diagnostic as a modifier of this data

A modifier diagnostic is a diagnostic that read this data and changes it in any way. The diagnostic must be a modifier even if it only affects the metadata

Parameters **diagnostic** (Diagnostic) –

clean_local()

Check if a local file is still needed and remove it if not

create_link()
Create a link from the original data in the <frequency>_<var_type> folder

dispatch(**args*)
Notify update to all the suscribers

Parameters **args** – arguments to pass

download()
Get data from the THREDDS server

Raises *THREDDSError* – If the data can not be downloaded

download_required()
Get if a download is required for this file

Returns

Return type bool

classmethod from_storage(*filepath*, *data_convention*)
Create a new datafile to be downloaded from the storage

has_modifiers()
Check if it has registered modifiers

Returns

Return type bool

local_status
Get local storage status

only_suscriber(*who*)
Get if an object is the sole suscriber of this publisher

Parameters **who** (*object*) –

Returns

Return type bool

prepare_to_upload(*rename_var*)
Prepare a local file to be uploaded

This includes renaming the variable if necessary, updating the metadata and adding the history and managing the possibility of multiple regions

ready_to_run(*diagnostic*)
Check if the data is ready to run for a given diagnostics

To be ready to run, the datafile should be in the local storage and no modifiers can be pending.

Parameters **diagnostic** (Diagnostic) –

Returns

Return type bool

set_local_file(*local_file*, *diagnostic=None*, *rename_var=''*, *region=None*)
Set the local file generated by EarthDiagnostics

This also prepares it for the upload

Parameters

- **local_file** (*str*) –

- **diagnostic** (*Diagnostic or None*) –
- **rename_var** (*str*) –
- **region** (*Basin or None*) –

Returns

Return type None

size
File size

storage_status
Get remote storage status

subscribe(*who*, *callback=None*)
Add a suscriber to the current publisher

Parameters

- **who** (*object*) – Subscriber to add
- **callback** (*callable or None, optional*) – Callback to call

suscribers
List of suscribers of this publisher

classmethod to_storage(*remote_file*, *data_convention*)
Create a new datafile object for a file that is going to be generated and stored

unsubscribe(*who*)
Remove a suscriber from the current publisher

Parameters **who** (*object*) – suscriber to remove

upload()
Send a loal file to the storage

upload_required()
Get if an upload is needed for this file

Returns

Return type bool

8.1.17 earthdiagnostics.utils

Common utilities for multiple topics that are not big enough to have their own module

class earthdiagnostics.utils.**TempFile**
Bases: object

Class to manage temporal files

autoclean = True
If True, new temporary files are added to the list for future cleaning

static clean()
Remove all temporary files created with Tempfile until now

files = []
List of files to clean automatically

static get (*filename=None*, *clean=None*, *suffix='.nc'*)
Get a new temporal filename, storing it for automated cleaning

Parameters

- **suffix** –
- **filename** (*str*) – if it is not none, the function will use this filename instead of a random one
- **clean** (*bool*) – if true, stores filename for cleaning

Returns path to the temporal file

Return type str

prefix = 'temp'
Prefix for temporary filenames

scratch_folder = ''
Scratch folder to create temporary files on it

class earthdiagnostics.utils.**Utils**
Bases: object

Container class for miscellaneous utility methods

exception CopyException
Bases: exceptions.Exception

Exception raised when copy fails

exception ExecutionError
Bases: exceptions.Exception

Exception to raise when a command execution fails

exception UnzipException
Bases: exceptions.Exception

Exception raised when unzip fails

static available_cpu_count()
Number of available virtual or physical CPUs on this system

cdo = <cdo.Cdo object>
An instance of Cdo class ready to be used

static check_netcdf_file (*filepath*)
Check if a NetCDF file is well stored

This functions is used to check if a NetCDF file is corrupted. It prefers to raise a false postive than to have false negatives.

Parameters **filepath** –

Returns

Return type bool

static concat_variables (*source*, *destiny*, *remove_source=False*)
Add variables from a nc file to another

Parameters

- **source** (*str*) –

- **destiny** (*str*) –
- **remove_source** (*bool*) – if True, removes source file

static convert2netcdf4 (*filetoconvert*)
Convert a file to NetCDF4

Conversion only performed if required. Deflation level set to 4 and shuffle activated.

Parameters **filetoconvert** (*str*) –

static convert_to_ascii_if_possible (*string*, *encoding='ascii'*)
Convert an Unicode string to ASCII if all characters can be translated.

If a string can not be translated it is unchanged. It also automatically replaces Bretonnière with Bretonniere

Parameters

- **string** (*unicode*) –
- **encoding** (*str, optional*) –

Returns

Return type str

static convert_units (*var_handler*, *new_units*, *calendar=None*, *old_calendar=None*)
Convert units

Parameters

- **var_handler** (*Dataset*) –
- **new_units** (*str*) –
- **calendar** (*str*) –
- **old_calendar** (*str*) –

static copy_attributes (*new_var*, *original_var*, *omitted_attributtes=None*)
Copy attributtes from one variable to another

Parameters

- **new_var** (*netCDF4.Variable*) –
- **original_var** (*netCDF4.Variable*) –
- **omitted_attributtes** (*iterable of str*) – Collection of attributtes that should not be copied

static copy_dimension (*source*, *destiny*, *dimension*, *must_exist=True*, *new_names=None*, *rename_dimension=False*)
Copy the given dimension from source to destiny, including dimension variables if present

Parameters

- **source** (*netCDF4.Dataset*) –
- **destiny** (*netCDF4.Dataset*) –
- **dimension** (*str*) –
- **must_exist** (*bool, optional*) –
- **new_names** (*dict of str: str or None, optional*) –

static copy_file (*source*, *destiny*, *save_hash=False*, *use_stored_hash=True*, *retrials=3*)
Copy a file and compute a hash to check if the copy is equal to the source

Parameters

- **source** (*str*) –
- **destiny** (*str*) –
- **save_hash** (*bool, optional*) – If True, stores a copy of the hash
- **use_stored_hash** (*bool, optional*) – If True, try to use the stored value of the source hash instead of computing it
- **retrials** (*int, optional*) – Minimum value is 1

See also:

move_file()

static copy_tree(*source*, *destiny*)
Copy a full tree to a new location

Parameters

- **source** (*str*) –
- **destiny** (*str*) –

See also:

move_tree()

static copy_variable(*source*, *destiny*, *variable*, *must_exist=True*, *add_dimensions=False*, *new_names=None*, *rename_dimension=True*)
Copy the given variable from source to destiny

Parameters

- **source** (*netCDF4.Dataset*) –
- **destiny** (*netCDF4.Dataset*) –
- **variable** (*str*) –
- **must_exist** (*bool, optional*) –
- **add_dimensions** (*bool, optional*) –
- **new_names** (*dict of str: str*) –

Raises Exception – If dimensions are not correct in the destiny file and add_dimensions is False

static create_folder_tree(*path*)
Create a folder path with all parent directories if needed.

Parameters **path** (*str*) –

static execute_shell_command(*command*, *log_level=10*)
Execute shell command

Writes the output to the log with the specified level

Parameters

- **command** (*str or iterable of str*) –
- **log_level** (*int, optional*) –

Returns Standard output of the command

Return type iterable of str

Raises *Utils.ExecutionError* – If the command return value is non zero

static get_datetime_from_netcdf(*handler*, *time_variable='time'*)
Get time from NetCDF files

Parameters

- **handler** (*netCDF4.Dataset*) –
- **time_variable** (*str, optional*) –

Returns

Return type numpy.array of Datetime

static get_file_hash(*filepath*, *use_stored=False*, *save=False*)
Get the xxHash hash for a given file

Parameters

- **filepath** (*str*) –
- **use_stored** (*bool, optional*) – If True, tries to use the stored hash before computing it
- **save** (*bool, optional*) – If True, saves the hash to a file

static get_file_variables(*filename*)
Get all the variables in a file

Parameters **filename** –

Returns

Return type iterable of str

static get_mask(*basin*, *with_levels=False*)
Return the mask for the given basin

Parameters **basin** (*Basin*) –

Returns

Return type numpy.array

Raises Exception: If mask.regions.nc is not available

static give_group_write_permissions(*path*)
Give write permissions to the group

static move_file(*source*, *destiny*, *save_hash=False*, *retrials=3*)
Move a file and compute a hash to check if the copy is equal to the source

It is just a call to Utils.copy_file followed bu

Parameters

- **source** (*str*) –
- **destiny** (*str*) –
- **save_hash** (*bool, optional*) – If True, stores a copy of the hash
- **retrials** (*int, optional*) – Minimum value is 1

See also:

copy_file()

static move_tree(*source*, *destiny*)
Move a tree to a new location

Parameters

- **source** (*str*) –
- **destiny** (*str*) –

See also:

copy_tree()

nco = <nco.nco.Nco object>
An instance of Nco class ready to be used

static open_cdf(*filepath*, *mode='a'*)
Open a NetCDF file

Parameters

- **filepath** (*str*) –
- **mode** (*str, optional*) –

Returns

Return type netCDF4.Dataset

static remove_file(*path*)
Delete a file only if it previously exists

Parameters **path** (*str*) –

static rename_variable(*filepath*, *old_name*, *new_name*, *must_exist=True*, *rename_dimension=True*)
Rename variable from a NetCDF file

This function is just a wrapper around Utils.rename_variables

Parameters

- **filepath** (*str*) –
- **old_name** (*str*) –
- **new_name** (*str*) –
- **must_exist** (*bool, optional*) –

See also:

Utils.rename_variables()

static rename_variables(*filepath*, *dic_names*, *must_exist=True*, *rename_dimension=True*)
Rename multiple variables from a NetCDF file

Parameters

- **filepath** (*str*) –
- **dic_names** (*dict of str: str*) – Gives the renaming to do in the form old_name: new_name
- **must_exist** (*bool, optional*) –

Raises

- ValueError – If any original name is the same as the new

- Exception – If any requested variable does not exist and must_exist is True

static setminmax(*filename*, *variable_list*)
Set the valid_max and valid_min values to the current max and min values on the file

Parameters

- **filename** (*str*) –
- **variable_list** (*str or iterable of str*) –

static untar(*files*, *destiny_path*)
Untar files to a given destiny

Parameters

- **files** (*iterable of str*) –
- **destiny_path** (*str*) –

static unzip(*files*, *force=False*)
Unzip a list of files

files: str or iterable of str force: bool, optional

if True, it will overwrite unzipped files

earthdiagnostics.utils.**suppress_stdout**(**args*, ***kwds*)
Redirect the standard output to devnull

8.1.18 earthdiagnostics.variable

Classes to manage variable definitions and aliases

class earthdiagnostics.variable.**CMORTable**(*name*, *frequency*, *date*, *domain*)
Bases: object

Class to represent a CMOR table

Parameters

- **name** (*str*) –
- **frequency** (Frequency) –
- **date** (*str*) –

class earthdiagnostics.variable.**Variable**
Bases: object

Class to characterize a CMOR variable.

It also contains the static method to make the match between the original name and the standard name. Requires data _convetion to be available in cmor_tables to work.

add_table(*table*, *priority=None*)
Add table to variable

Parameters

- **table** (CMORTable) –
- **priority** (*int or None, optional*) –

get_modelling_realm(*domains*)
Get var modelling realm

Parameters **domains** (*iterable of str*) –

Returns

Return type ModelingRealm or None

get_table(*frequency*, *data_convention*)
Get a table object given the frequency and data_covention

If the variable does not contain the table information, it uses the domain to make a guess

Parameters

- **frequency** (Frequency) –
- **data_convention** (*str*) –

Returns

Return type *CMORTable*

Raises ValueError – If a table can not be deduced from the given parameters

parse_csv(*var_line*)
Fill the object information from a csv line

Parameters **var_line** (*list of str*) –

parse_json(*json_var*, *variable*)
Parse variable json

Parameters

- **json_var** (*dict of str: str*) –
- **variable** (*str*) –

class earthdiagnostics.variable.**VariableAlias**(*alias*, *basin=None*, *grid=None*)
Bases: object

Class to characterize a CMOR variable.

It also contains the static method to make the match between the original name and the standard name. Requires data _convetion to be available in cmor_tables to work.

Parameters **alias** (*str*) –

exception earthdiagnostics.variable.**VariableJsonException**
Bases: exceptions.Exception

Exception to be raised when an error related to the json reading is encountered

class earthdiagnostics.variable.**VariableManager**
Bases: object

Class for translating variable alias into standard names and provide the correct description for them

clean()
Clean all information contained in the variable manager

create_aliases_dict()
Create aliases dictionary for the registered variables

get_all_variables()
Return all variables

Returns CMOR variable list

Return type set[*Variable*]

get_variable(*original_name*, *silent=False*)
Return the cmor variable instance given a variable name

Parameters

- **original_name** (*str*) – original variable's name
- **silent** (*bool*) – if True, omits log warning when variable is not found

Returns CMOR variable

Return type *Variable*

get_variable_and_alias(*original_name*, *silent=False*)
Return the cmor variable instance given a variable name

Parameters

- **original_name** (*str*) – original variable's name
- **silent** (*bool*) – if True, omits log warning when variable is not found

Returns CMOR variable

Return type *Variable*

load_variables(*table_name*)
Load the CMOR csv and creates the variables dictionary

Parameters **table_name** (*str*) –

register_variable(*var*)
Register variable info

Parameters **var** (Variable) –

class earthdiagnostics.variable.**VariableType**
Bases: object

Enumeration of variable types

static to_str(*vartype*)
Get str representation of vartype for the folder convention

8.1.19 earthdiagnostics.variable_type

8.1.20 earthdiagnostics.workmanager

Earthdiagnostics workflow manager

class earthdiagnostics.work_manager.**Downloader**
Bases: object

Download manager for EarthDiagnostics

We are not using a ThreadPoolExecutor because we want to be able to control priorities in the download

shutdown()
Stop the downloader after all downloads have finished

start()
Create the downloader thread and initialize it

submit (*datafile*)
Add a datafile to the download queue

class earthdiagnostics.work_manager.**WorkManager** (*config*, *data_manager*)
Bases: object

Class to produce and control the workflow of EarthDiagnostics

Parameters

- **config** (Config) –
- **data_manager** (DataManager) –

prepare_job_list ()
Create the list of jobs to run

run ()
Run all the diagnostics

Returns Only True if all diagnostic were correctly executed

Return type bool

8.2 earthdiagnostics.general

8.2.1 earthdiagnostics.general.attribute

Set attributtes in netCDF files

class earthdiagnostics.general.attribute.**Attribute** (*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *grid*, *attributte_name*, *attributte_value*)
Bases: earthdiagnostics.general.fix_file.FixFile

Set the value of an attribute

Can be useful to correct wrong metadata

Original author Javier Vegas-Regidor<javier.vegas@bsc.es>

Created July 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable's name
- **domain** (*ModelingRealm*) – variable's domain

alias = 'att'
Diagnostic alias for the configuration file

compute ()
Run the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, domain, grid

Returns

8.2.2 earthdiagnostics.general.dailymean

8.2.3 earthdiagnostics.general.module

Compute module of two variables

class earthdiagnostics.general.module.**Module**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *componentu*, *componentv*, *module_var*, *grid*)
Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute the module of the vector given by two scalar variables

Original author Javier Vegas-Regidor<javier.vegas@bsc.es>

Created July 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int :*) – chunk's number
- **domain** (*ModelingRealm*) – variable's domain

alias = 'module'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, domain, grid

Returns

request_data()
Request data required by the diagnostic

8.2.4 earthdiagnostics.general.monthlymean

8.2.5 earthdiagnostics.general.relink

Create links for a variable

class earthdiagnostics.general.relink.**Relink**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *move_old*, *grid*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Recreates the links for the variable specified

Original author Javier Vegas-Regidor<javier.vegas@bsc.es>

Created September 2016

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable's name
- **domain** (*ModelingRealm*) – variable's domain
- **move_old** (*bool*) – if true, looks for files following the old convention and moves to avoid collisions

alias = 'relink'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, domain, move_old=False

Returns

request_data()
Request data required by the diagnostic

8.2.6 earthdiagnostics.general.relinkall

Create links for all variables in a startdate

class earthdiagnostics.general.relinkall.**RelinkAll**(*data_manager*, *startdate*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Recreates the links for the variable specified

Original author Javier Vegas-Regidor<javier.vegas@bsc.es>

Created September 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate

alias = 'relinkall'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, domain, move_old=False

Returns

request_data()
Request data required by the diagnostic

8.2.7 earthdiagnostics.general.rewrite

Rewrite netCDF file

class earthdiagnostics.general.rewrite.**Rewrite**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *grid*)
Bases: earthdiagnostics.general.fix_file.FixFile

Rewrites files without doing any calculations.

Can be useful to convert units or to correct wrong metadata

Original author Javier Vegas-Regidor<javier.vegas@bsc.es>

Created July 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable's name
- **domain** (*ModelingRealm*) – variable's domain

alias = 'rewrite'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

8.2.8 earthdiagnostics.general.scale

Scales a variable by with value and offset

class earthdiagnostics.general.scale.**Scale**(*data_manager*, *startdate*, *member*, *chunk*, *value*, *offset*, *domain*, *variable*, *grid*, *min_limit*, *max_limit*, *frequency*, *apply_mask*)
Bases: earthdiagnostics.general.fix_file.FixFile

Scales a variable by the given value also adding at offset

Can be useful to correct units or other known errors (think of a tas file declaring K as units but with the data stored as Celsius)

Original author Javier Vegas-Regidor<javier.vegas@bsc.es>

Created July 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int :*) – chunk's number
- **variable** (*str*) – variable's name
- **domain** (*ModelingRealm*) – variable's domain

alias = 'scale'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, domain, grid

Returns

8.2.9 earthdiagnostics.general.simplify_dimensions

Convert i j files to lon lat when there is no interpolation required

class earthdiagnostics.general.simplify_dimensions.**SimplifyDimensions**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *grid*, *data_convention*)

Bases: earthdiagnostics.general.fix_file.FixFile

Convert i j files to lon lat when there is no interpolation required

i.e. lon is constant over i and lat is constat over j

Parameters

- **data_manager** (DataManager) –
- **startdate** (*str*) –
- **member** (*int*) –
- **chunk** (*init*) –
- **domain** (*ModellingRealm*) –
- **variable** (*str*) –
- **grid** (*str or None*) –
- **data_convention** (*str*) –

alias = 'simdim'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – domain,variables,grid

Returns

8.2.10 earthdiagnostics.general.yearlymean

8.3 earthdiagnostics.ocean

8.3.1 earthdiagnostics.ocean.areamoc

Compute an Atlantic MOC index from the average

class earthdiagnostics.ocean.areamoc.**AreaMoc**(*data_manager*, *startdate*, *member*, *chunk*, *basin*, *box*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute an Atlantic MOC index

Averages the meridional overturning in a latitude band between 1km and 2km or any other index averaging the meridional overturning in a given basin and a given domain

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created March 2012

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **basin** (Basin) – basin to compute
- **box** (Box) – box to compute

alias = 'mocarea'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – minimum latitude, maximum latitude, minimum depth, maximum depth, basin=Global

Returns

request_data()
Request data required by the diagnostic

8.3.2 earthdiagnostics.ocean.averagesection

Compute an average of a given zone

class earthdiagnostics.ocean.averagesection.**AverageSection**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *box*, *grid*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute an average of a given zone.

The variable MUST be in a regular grid

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created March 2012

Last modified June 2016

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable's name
- **domain** (*ModelingRealm*) – variable's domain
- **box** (*Box*) – box to use for the average

alias = 'avgsection'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, minimum longitude, maximum longitude, minimum latitude, maximum latitude, domain=ocean

Returns

request_data()
Request data required by the diagnostic

8.3.3 earthdiagnostics.ocean.convectionsites

Compute the intensity of convection

class earthdiagnostics.ocean.convectionsites.**ConvectionSites**(*data_manager*, *startdate*, *member*, *chunk*, *model_version*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute the intensity of convection in the four main convection sites

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created October 2013

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **model_version** (*str*) – model version

alias = 'convection'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – None

Returns

request_data()
Request data required by the diagnostic

8.3.4 earthdiagnostics.ocean.cutsection

Cut meridional or zonal sections

class earthdiagnostics.ocean.cutsection.**CutSection**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *zonal*, *value*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Cuts a meridional or zonal section

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created September 2012

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number

- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable's name
- **domain** (*Domain*) – variable's domain
- **zonal** (*bool*) – specifies if section is zonal or meridional
- **value** (*int*) – value of the section's coordinate

alias = 'cutsection'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, zonal, value, domain=ocean

Returns

request_data()
Request data required by the diagnostic

8.3.5 earthdiagnostics.ocean.gyres

Compute the intensity of the subtropical and subpolar gyres

class earthdiagnostics.ocean.gyres.**Gyres**(*data_manager*, *startdate*, *member*, *chunk*, *model_version*)
Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute the intensity of the subtropical and subpolar gyres

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created October 2013

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **model_version** (*str*) – model version

alias = 'gyres'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – None

Returns

request_data()
Request data required by the diagnostic

8.3.6 earthdiagnostics.ocean.heatcontent

Compute the total ocean heat content

class earthdiagnostics.ocean.heatcontent.**HeatContent**(*data_manager*, *startdate*, *member*, *chunk*, *basin*, *mixed_layer*, *box*, *min_level*, *max_level*)
Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute the total ocean heat content

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created May 2012

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **mixed_layer** (*int*) – If 1, restricts calculation to the mixed layer, if -1 exclude it. If 0, no effect
- **box** (Box) – box to use for the average

alias = 'ohc'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – basin, mixed layer option (1 to only compute at the mixed layer, -1 to exclude it, 0 to ignore), minimum depth, maximum depth

Returns

request_data()
Request data required by the diagnostic

8.3.7 earthdiagnostics.ocean.heatcontentlayer

Point-wise Ocean Heat Content in a specified ocean thickness (J/m-2)

class earthdiagnostics.ocean.heatcontentlayer.**HeatContentLayer**(*data_manager*, *startdate*, *member*, *chunk*, *box*, *weight*, *min_level*, *max_level*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Point-wise Ocean Heat Content in a specified ocean thickness (J/m-2)

Original author Isabel Andreu Burillo

Contributor Virginie Guemas <virginie.guemas@bsc.es>

Contributor Eleftheria Exarchou <eleftheria.exarchou@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created June 2012

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **box** (Box) – box to use for the calculations

alias = 'ohclayer'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class

- **options** (*list[str]*) – minimum depth, maximum depth, basin=Global

request_data()
Request data required by the diagnostic

8.3.8 earthdiagnostics.ocean.interpolate

SCRIP based interpolation

class earthdiagnostics.ocean.interpolate.**Interpolate**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *target_grid*, *model_version*, *invert_lat*, *original_grid*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

3-dimensional conservative interpolation to the regular atmospheric grid.

It can also be used for 2D (i,j) variables

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created November 2012

Last modified June 2016

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable's name
- **domain** (*Domain*) – variable's domain
- **model_version** (*str*) – model version

alias = 'interp'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – target_grid, variable, domain=ocean

Returns

request_data()
Request data required by the diagnostic

8.3.9 earthdiagnostics.ocean.interpolatecdo

CDO-based interpolation

class earthdiagnostics.ocean.interpolatecdo.**ComputeWeights**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *target_grid*, *original_grid*, *weights_file*, *method*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Diagnostic used to compute interpolation weights

Parameters

- **data_manager** (*DataManager*) –
- **startdate** (*str*) –
- **member** (*int*) –
- **chunk** (*int*) –
- **domain** (*ModelingRealm*) –
- **variable** (*str*) –
- **target_grid** (*str*) –
- **original_grid** (*str*) –
- **weights_file** (*str*) –
- **method** (*str*) –

alias = 'computeinterpcdoweights'
Diagnostic alias for the configuration file

compute()
Compute weights

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Generate the instances of the diagnostics that will be run by the manager

This method does not does anything as this diagnostic is not expected to be called by the users

request_data()
Request data required by the diagnostic

class earthdiagnostics.ocean.interpolatecdo.**InterpolateCDO**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *target_grid*, *model_version*, *mask_oceans*, *original_grid*, *weights*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

3-dimensional conservative interpolation to the regular atmospheric grid.

It can also be used for 2D (i,j) variables

Original author Javier Vegas-Regidor<javier.vegas@bsc.es>

Created October 2016

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable's name
- **domain** (*ModelingRealm*) – variable's domain
- **model_version** (*str*) – model version

alias = 'interpcdo'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

classmethod compute_weights(*method*, *target_grid*, *sample_file*, *weights*)
Compute weights for interpolation from sample file

Parameters

- **method** (*int*) – Interpolation method
- **target_grid** (*str*) – Grid to intepolate to. Can be anything understand by CDO
- **sample_file** (*str*) – Path to a file containing original mesh information
- **weights** – Path to the file to store the weights

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – target_grid, variable, domain=ocean

Returns

classmethod get_sample_grid_file()
Get a sample grid file

Create a sample grid file from the definition in the masks file

Returns

Return type str

request_data()
Request data required by the diagnostic

8.3.10 earthdiagnostics.ocean.maskland

8.3.11 earthdiagnostics.ocean.maxmoc

Compute an Atlantic MOC index

class earthdiagnostics.ocean.maxmoc.**MaxMoc**(*data_manager*, *startdate*, *member*, *year*, *basin*, *box*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute an Atlantic MOC index

It finds the maximum of the annual mean meridional overturning in a latitude / depth region

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created March 2012

Last modified June 2016

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **year** (*int*) – year to compute
- **basin** (*Basin*) – basin to compute
- **box** (*Box*) – box to compute

alias = 'mocmax'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each complete year to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – minimum latitude, maximum latitude, minimum depth, maximum depth, basin=global

Returns

request_data()
Request data required by the diagnostic

8.3.12 earthdiagnostics.ocean.mixedlayerheatcontent

Compute mixed layer heat content

class earthdiagnostics.ocean.mixedlayerheatcontent.**MixedLayerHeatContent**(*data_manager*, *startdate*, *member*, *chunk*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute mixed layer heat content

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created February 2012

Last modified June 2016

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number

alias = 'mlotsthc'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – None

Returns

request_data()
Request data required by the diagnostic

8.3.13 earthdiagnostics.ocean.mixedlayersaltcontent

Compute mixed layer salt content

class earthdiagnostics.ocean.mixedlayersaltcontent.**MixedLayerSaltContent**(*data_manager*, *startdate*, *member*, *chunk*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute mixed layer salt content

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created February 2012

Last modified June 2016

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number

alias = 'mlotstsc'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – None

Returns

request_data()
Request data required by the diagnostic

8.3.14 earthdiagnostics.ocean.moc

Compute the MOC for oceanic basins

class earthdiagnostics.ocean.moc.**Moc**(*data_manager*, *startdate*, *member*, *chunk*)
Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute the MOC for oceanic basins

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created March 2012

Last modified June 2016

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number

alias = 'moc'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – None

Returns

request_data()
Request data required by the diagnostic

8.3.15 earthdiagnostics.ocean.mxl

Compute the mixed layer depth

class earthdiagnostics.ocean.mxl.**Mxl**(*data_manager*, *startdate*, *member*, *chunk*)
Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute the mixed layer depth

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number

alias = 'mxl'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – None

Returns

request_data()
Request data required by the diagnostic

8.3.16 earthdiagnostics.ocean.psi

Compute the barotropic stream function

class earthdiagnostics.ocean.psi.**Psi**(*data_manager*, *startdate*, *member*, *chunk*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute the barotropic stream function

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created March 2012

Last modified June 2016

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number

alias = 'psi'

Diagnostic alias for the configuration file

compute()

Run the diagnostic

declare_data_generated()

Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)

Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – None

Returns

request_data()

Request data required by the diagnostic

8.3.17 earthdiagnostics.ocean.rotation

Rotate two u v variables to align with latitude and longitude

class earthdiagnostics.ocean.rotation.**Rotation**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variableu*, *variablev*, *executable*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Rotate two u v variables to align with latitude and longitude

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created September 2012

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **domain** (*Domain*) – variable's domain

alias = 'rotate'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, zonal, value, domain=ocean

Returns

request_data()
Request data required by the diagnostic

8.3.18 earthdiagnostics.ocean.siasiesiv

Compute the sea ice extent , area and volume in both hemispheres or a specified region

class earthdiagnostics.ocean.siasiesiv.**Siasiesiv**(*data_manager*, *startdate*, *member*, *chunk*, *masks*, *var_manager*, *data_convention*, *omit_vol*)
Bases: *earthdiagnostics.diagnostic.Diagnostic*

Compute the sea ice extent , area and volume in both hemispheres or a specified region.

Parameters

- **data_manager** (DataManager) –
- **startdate** (*str*) –
- **member** (*int*) –
- **chunk** (*init*) –
- **domain** (*ModellingRealm*) –
- **variable** (*str*) –
- **basin** (*list of Basin*) –
- **mask** (*numpy.array*) –
- **omit_vol** (*bool*) –

alias = 'siasiesiv'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – basin

Returns

request_data()
Request data required by the diagnostic

8.3.19 earthdiagnostics.ocean.verticalgradient

Calculate the gradient between 2 ocean levels

class earthdiagnostics.ocean.verticalgradient.**VerticalGradient**(*data_manager*, *startdate*, *member*, *chunk*, *variable*, *box*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Calculate the gradient between 2 ocean levels

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Eleftheria Exarchou <eleftheria.exarchou@bsc.es>

Contributor Javier Vegas-Regidor <javier.vegas@bsc.es>

Created February 2012

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable to average
- **box** (Box) – box used to restrict the vertical mean

alias = 'vgrad'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, minimum depth (level), maximum depth (level)

Returns

request_data()
Request data required by the diagnostic

8.3.20 earthdiagnostics.ocean.verticalmean

Chooses vertical level in ocean, or vertically averages between 2 or more ocean levels

class earthdiagnostics.ocean.verticalmean.**VerticalMean**(*data_manager*, *startdate*, *member*, *chunk*, *variable*, *box*)
Bases: *earthdiagnostics.diagnostic.Diagnostic*

Chooses vertical level in ocean, or vertically averages between 2 or more ocean levels

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Eleftheria Exarchou <eleftheria.exarchou@bsc.es>

Contributor Javier Vegas-Regidor <javier.vegas@bsc.es>

Created February 2012

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable to average
- **box** (Box) – box used to restrict the vertical mean

alias = 'vertmean'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, minimum depth (level), maximum depth (level)

Returns

request_data()
Request data required by the diagnostic

8.3.21 earthdiagnostics.ocean.verticalmeanmeters

Averages vertically any given variable

class earthdiagnostics.ocean.verticalmeanmeters.**VerticalMeanMeters**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *box*, *grid_point*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Averages vertically any given variable

Original author Virginie Guemas <virginie.guemas@bsc.es>

Contributor Javier Vegas-Regidor<javier.vegas@bsc.es>

Created February 2012

Last modified June 2016

Parameters

- **data_manager** (DataManager) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable to average
- **box** (Box) – box used to restrict the vertical mean

alias = 'vertmeanmeters'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, minimum depth (meters), maximum depth (meters)

Returns

request_data()
Request data required by the diagnostic

8.4 earthdiagnostics.statistics

8.4.1 earthdiagnostics.statistics.climatologicalpercentile

Calculates the climatological percentiles for the given leadtime

class earthdiagnostics.statistics.climatologicalpercentile.**ClimatologicalPercentile**(*data_manager*, *domain*, *variable*, *start_year*, *end_year*, *forecast_month*, *experiment_config*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Calculates the climatological percentiles for the given leadtime

Parameters

- **data_manager** (DataManager) – data management object
- **variable** (*str*) – variable to average
- **experiment_config** (ExperimentConfig) –

alias = 'climpercent'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – domain, variable, percentil number, maximum depth (level)

Returns

request_data()
Request data required by the diagnostic

requested_startdates()
Required startdates to compute the percentile

Returns

Return type list of str

8.4.2 earthdiagnostics.statistics.monthlypercentile

Calculates the montlhy percentiles

class earthdiagnostics.statistics.monthlypercentile.**MonthlyPercentile**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *percentiles*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Calculates the montlhy percentiles

Parameters

- **data_manager** (*DataManager*) – data management object
- **startdate** (*str*) – startdate
- **member** (*int*) – member number
- **chunk** (*int*) – chunk's number
- **variable** (*str*) – variable to average

alias = 'monpercent'
Diagnostic alias for the configuration file

compute()
Run the diagnostic

declare_data_generated()
Declare data to be generated by the diagnostic

classmethod generate_jobs(*diags*, *options*)
Create a job for each chunk to compute the diagnostic

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – domain, variable, percentil number, maximum depth (level)

Returns

percentile(*percentile*)
Variable name for the given percentile

Parameters **percentile** (*int*) –

Returns

Return type str

request_data()
Request data required by the diagnostic

variable_max
Variable name for the maximum

Returns

Return type str

variable_min
Variable name for the minimum

Returns

Return type str

Python Module Index

e

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W