Earth Diagnostics Documentation

Release 3.0.0b28

BSC-CNS Earth Sciences Department

Jan 11, 2017

CONTENTS

CHAPTER

ONE

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

For now, you only have one option: download the diagnostics directly from BSC-ES's Gitlab:

```
git clone https://earth.bsc.es/gitlab/es/ocean_diagnostics.git
```

You will also need

- CDO version 1.6.9 (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 Autosubmit, CDO and NCO packages, among others. A virtual environment with all requisites fullfilled is available at /shared/earth/ClimatePrediction/EarthDiagnostics
- Access to CDFTOOLS_3.0 executables for BSC-ES. At this point, those are located at /shared/earth/ClimatePrediction/CDFTOOLS_CMOR/bin.

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, a diags.conf file that can be used as a model 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 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,$VARIABLE,$DOMAIN
```

1.3 Prepare the run script

Once you have configured your experiment you can execute any diagnostic with the provided launch_diags.sh script. Create a copy and change the variables PATH_TO_CONF_FILE and PATH_TO_DIAGNOSTICS so they point to your conf file and installation folder.

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

TWO

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
```

2.1 General

- **att:** Writes a global attributte to all the netCDF files. See *Attribute*
- **monmean:** Calculates the monthly mean of the given variable. See *MonthlyMean*
- **relink:** Regenerates the links created in the monthly_mean, daily_mean, folders. See *Relink*
- **rewrite:** Just rewrites the CMOR output of a given variable. Useful to correct metadata or variable units. See *Rewrite*
- **scale:** Scales a given variable using a given scale factor and offset. Useful to correct erros on the data. See *Scale*

2.2 Ocean

- **areamoc:** Compute an Atlantic MOC index. See *AreaMoc*
- **averagesection:** Compute an average of a given zone. The variable MUST be in a regular grid See *AverageSection*
- **convectionsites:** Compute the intensity of convection in the four main convection sites. See *ConvectionSites*
- **cutsection:** Cuts a meridional or zonal section. See *CutSection*
- **gyres:** Compute the intensity of the subtropical and subpolar gyres. See *Gyres*
- **heatcontent:** Compute the total ocean heat content. See *HeatContent*
- **heatcontentlayer:** Point-wise Ocean Heat Content in a specified ocean thickness. See *HeatContentLayer*
- **interpolate:** 3-dimensional conservative interpolation to the regular atmospheric grid. It can also be used for 2D (i,j) variables. See *Interpolate*
- **interpolateCDO:** Bilinear interpolation to a given grid using CDO. See *InterpolateCDO*

- **maxmoc:** Compute an Atlantic MOC index by finding the maximum of the annual mean meridional overturning in a latitude / depth region See *MaxMoc*
- **mixedlayerheatcontent:** Compute mixed layer heat content. See *MixedLayerHeatContent*
- **mixedlayersaltcontent:** Compute mixed layer salt content. See *MixedLayerSaltContent*
- **moc:** Compute the MOC for oceanic basins. See *Moc*
- **psi:** Compute the barotropic stream function. See *Psi*
- **siasiesiv:** Compute the sea ice extent , area and volume in both hemispheres or a specified region. See *Siasiesiv*
- **verticalmean:** Chooses vertical level in ocean, or vertically averages between 2 or more ocean levels. See *VerticalMean*
- **verticalmeanmeters:** Averages vertically any given variable. See *VerticalMeanMeters*

2.3 Statistics

- **climpercent:** Calculates the specified climatological percentile of a given variable. See *ClimatologicalPercentile*
- **monpercent:** Calculates the specified monthly percentile of a given variable. See *MonthlyPercentile*

CHAPTER

THREE

TIPS AND TRICKS

3.1 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.

3.2 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.

3.3 NEMO files

Unlike the bash version of the ocean diagnostics, this program keeps the NEMO files in the scratch folder so you can launch different configurations for the same experiment with reduced start time. You will need to remove the experiment's folder in the scratch directory at the end of the experiment to avoid wasting resources.

CHAPTER

FOUR

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

FIVE

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.

5.1 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

SIX

FREQUENTLY ASKED QUESTIONS

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

CHAPTER

SEVEN

MODULE DOCUMENTATION

7.1 earthdiagnostics

7.1.1 earthdiagnostics.box

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

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()
Gets a string representation of depth. For depth expressed in meters, it adds th character 'm' to the end If min_depth is different from max_depth, it concatenates the two values :return: string representation for depth :rtype: str

get_lat_str()
Gets 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 :return: string representation for latitude :rtype: str

get_lon_str()
Gets 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 :return: string representation for longitude :rtype: str

max_depth = None
Maximum depth :rtype: float

max_lat
Maximum latitude :rtype: float

max_lon
Maximum longitude :rtype: float

min_depth = None
Minimum depth :rtype: float

min_lat
Minimum latitude :rtype: float

min_lon
Minimum longitude :rtype: float

7.1.2 earthdiagnostics.cdftools

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

Class to run CDFTools executables

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

run(*command*, *input*, *output=None*, *options=None*, *log_level=20*)
Runs one of the CDFTools

Parameters

- **command** (*str | iterable*) – executable to run
- **input** (*str*) – input file
- **output** – output file. Not all tools support this parameter
- **options** (*str | list[str] | Tuple[str]*) – options for the tool.
- **log_level** (*int*) – log level at which the output of the cdftool command will be added

7.1.3 earthdiagnostics.cmorizer

class earthdiagnostics.cmorizer.**Cmorizer**(*data_manager*, *startdate*, *member*)
Bases: object

Class to manage CMORization

Parameters

- **data_manager** (CMORManager) – experiment's data manager
- **startdate** (*str*) – startdate to cmorize
- **member** (*int*) – member to cmorize

cmorize_atmos()
CMORizes atmospheric data, from grib or MMA files :return:

cmorize_ocean()
CMORizes ocean files from MMO files :return:

extract_variable(*file_path*, *handler*, *frequency*, *variable*)
Extracts a variable from a file and creates the CMOR file

Parameters

- **file_path** (*str*) – path to the file
- **handler** (*netCDF4.Dataset*) – netCDF4 handler for the file
- **frequency** (*Frequency*) – variable's frequency
- **variable** (*str*) – variable's name

7.1.4 earthdiagnostics.cmormanager

class earthdiagnostics.cmormanager.**CMORManager**(*config*)
Bases: *earthdiagnostics.datamanager.DataManager*

Data manager class for CMORized experiments

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

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

Parameters

- **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|NoneType*) – 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|NoneType*) – 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

get_file_path(*startdate*, *member*, *domain*, *var*, *chunk*, *frequency*, *box=None*, *grid=None*, *year=None*, *date_str=None*)

Returns the path to a concrete file :param startdate: file's startdate :type startdate: str :param member: file's member :type member: int :param domain: file's domain :type domain: Domain :param var: file's var :type var: var :param chunk: file's chunk :type chunk: int :param frequency: file's frequency :type frequency: Frequency :param box: file's box :type box: Box :param grid: file's grid :type grid: str|NoneType :param year: file's year :type year: int|str|NoneType :param date_str: date string to add directly. Overrides year or chunk configurations :type date_str: str|NoneType :return: path to the file :rtype: str|NoneType

get_year(*domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*)

Ge a file containing all the data for one year for one variable :param domain: variable's domain :type domain: str :param var: variable's name :type var: str :param startdate: startdate to retrieve :type startdate: str :param member: member to retrieve :type member: int :param year: year to retrieve :type year: int :param grid: variable's grid :type grid: str :param box: variable's box :type box: Box :return:

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

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

Parameters

- **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)
- **box** (Box) – file's box (only needed to retrieve sections or averages)
- **frequency** (*Frequency*) – 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()

Prepares the data to be used by the diagnostic.

If CMOR data is not created, it show a warning and closes. In the future, an automatic cmorization procedure will be launched

If CMOR data is available but packed, the procedure will unpack it.

Returns

send_file(*filetosend*, *domain*, *var*, *startdate*, *member*, *chunk=None*, *grid=None*, *region=None*, *box=None*, *rename_var=None*, *frequency=None*, *year=None*, *date_str=None*, *move_old=False*, *diagnostic=None*, *cmorized=False*, *vartype=1*)

Copies a given file to the CMOR repository. It also automatically converts to netCDF 4 if needed and can merge with already existing ones as needed

Parameters

- **move_old** (*bool*) – if true, moves files following older conventions that may be found on the links folder
- **date_str** – exact date_str to use in the cmorized file
- **year** (*int*) – if frequency is yearly, this parameter is used to give the corresponding year
- **rename_var** (*str*) – if exists, the given variable will be renamed to the one given by var
- **filetosend** (*str*) – path to the file to send to the CMOR repository
- **region** (*str*) – specifies the region represented by the file. If it is defined, the data will be appended to the CMOR repository as a new region in the file or will overwrite if region was already present
- **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)
- **box** (Box) – file's box (only needed to retrieve sections or averages)
- **frequency** (*Frequency*) – file's frequency (only needed if it is different from the default)
- **diagnostic** (Diagnostic) – diagnostic used to generate the file
- **cmorized** (*bool*) – flag to indicate if file was generated in cmorization process

- **vartype** (*VariableType*) – Variable type (mean, statistic)

Type str

7.1.5 earthdiagnostics.config

class earthdiagnostics.config.**Config**(*path*)
Bases: object

Class to read and manage the configuration

Parameters **path** (*str*) – path to the conf file

cdftools_path = None
Path to CDFTOOLS executables

con_files = None
Mask and meshes folder path

data_adaptor = None
Scratch folder path

data_dir = None
Root data folder path

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

experiment = None
Configuration related to the experiment

Return type *ExperimentConfig*

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

get_commands()
Returns the list of commands after replacing the alias :return: full list of commands :rtype: list(str)

max_cores = None
Maximum number of cores to use

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

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

Encapsulates all chunk related tasks

Parameters **parser** (Parser) – parser for the config file

get_chunk_list()
Return a list with all the chunks :return: List containing tuples of startdate, member and chunk :rtype: tuple[str, int, int]

get_full_years(*startdate*)
Returns the list of full years that are in the given startdate :param startdate: startdate to use :type startdate: str :return: list of full years :rtype: list[int]

get_member_list()
Return a list with all the members :return: List containing tuples of startdate and member :rtype: tuple[str, int, int]

get_member_str(*member*)
Returns the member name for a given member number. :param member: member's number :type member: int :return: member's name :rtype: str

get_year_chunks(*startdate*, *year*)
Get the list of chunks containing timesteps from the given year :param startdate: startdate to use :type startdate: str :param year: reference year :type year: int :return: list of chunks containing data from the given year :rtype: list[int]

7.1.6 earthdiagnostics.constants

Contains the enumeration-like classes used by the diagnostics

class earthdiagnostics.constants.**Basin**(*shortname*, *fullname*, *box=None*)
Bases: object

Class representing a given basin

Parameters

- **shortname** (*str*) – sfull basin's name
- **fullname** (*str*) – full basin's name
- **box** (Box) – box defining the basin

box = None
Box representing the basin

fullname
Basin's full name :rtype: str

shortname
Basin's short name :rtype: str

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

Predefined basins

Antarctic = <earthdiagnostics.constants.Basin object>
Antarctic Ocean

AntarcticAtlantic = <earthdiagnostics.constants.Basin object>
Antarctic Ocean Atlantic Sector

AntarcticIndian = <earthdiagnostics.constants.Basin object>
Antarctic Ocean Indian Sector

Arctic = <earthdiagnostics.constants.Basin object>
Arctic Ocean

ArcticMarginalSeas = <earthdiagnostics.constants.Basin object>
Arctic Ocean

ArcticNorthAtlantic = <earthdiagnostics.constants.Basin object>
Arctic Ocean North Atlantic

Atlantic = <earthdiagnostics.constants.Basin object>
Atlantic ocean

Baffin = <earthdiagnostics.constants.Basin object>
Baffin

Baffin_Bay = <earthdiagnostics.constants.Basin object>
Baffin_Bay

Baltic_Sea = <earthdiagnostics.constants.Basin object>
Baltic_Sea

BarKara = <earthdiagnostics.constants.Basin object>
BarKara

Barents_Sea = <earthdiagnostics.constants.Basin object>
Barents_Sea

Beaufort_Chukchi_Sea = <earthdiagnostics.constants.Basin object>
Beaufort_Chukchi_Sea

Beaufort_Sea = <earthdiagnostics.constants.Basin object>
Beaufort_Sea

Bellingshausen_Sea = <earthdiagnostics.constants.Basin object>
Bellingshausen_Sea

Bering = <earthdiagnostics.constants.Basin object>
Bering

Bering_Strait = <earthdiagnostics.constants.Basin object>
Bering_Strait

CanArch = <earthdiagnostics.constants.Basin object>
CanArch

Canadian_Waters = <earthdiagnostics.constants.Basin object>
Canadian_Waters

Caspian_Sea = <earthdiagnostics.constants.Basin object>
Caspian_Sea

Central_Arctic = <earthdiagnostics.constants.Basin object>
Central_Arctic

Chukchi_Sea = <earthdiagnostics.constants.Basin object>
Chukchi_Sea

East_Siberian_Sea = <earthdiagnostics.constants.Basin object>
East_Siberian_Sea

Eastern_Central_Arctic = <earthdiagnostics.constants.Basin object>
Eastern_Central_Arctic

Fram_Strait = <earthdiagnostics.constants.Basin object>
Fram_Strait

Global = <earthdiagnostics.constants.Basin object>
Global ocean

Global_Ocean = <earthdiagnostics.constants.Basin object>
Global_Ocean

Greenland_Sea = <earthdiagnostics.constants.Basin object>
Greenland_Sea

Grnland = <earthdiagnostics.constants.Basin object>
Grnland

Hudson = <earthdiagnostics.constants.Basin object>
Hudson

Icelandic_Sea = <earthdiagnostics.constants.Basin object>
Icelandic_Sea

Indian = <earthdiagnostics.constants.Basin object>
Indian Ocean

IndoPacific = <earthdiagnostics.constants.Basin object>
Indo Pacific Ocean

Irminger_Sea = <earthdiagnostics.constants.Basin object>
Irminger_Sea

Kara_Gate_Strait = <earthdiagnostics.constants.Basin object>
Kara_Gate_Strait

Kara_Sea = <earthdiagnostics.constants.Basin object>
Kara_Sea

Labrador_Sea = <earthdiagnostics.constants.Basin object>
Labrador_Sea

Laptev_East_Siberian_Chukchi_Seas = <earthdiagnostics.constants.Basin object>
Laptev_East_Siberian_Chukchi_Seas

Laptev_East_Siberian_Seas = <earthdiagnostics.constants.Basin object>
Laptev_East_Siberian_Seas

Laptev_Sea = <earthdiagnostics.constants.Basin object>
Laptev_Sea

Lincoln_Sea = <earthdiagnostics.constants.Basin object>
Lincoln_Sea

Mediterranean_Sea = <earthdiagnostics.constants.Basin object>
Mediterranean_Sea

Nares_Strait = <earthdiagnostics.constants.Basin object>
Nares_Strait

Nordic_Barents_Seas = <earthdiagnostics.constants.Basin object>
Nordic_Barents_Seas

Nordic_Seas = <earthdiagnostics.constants.Basin object>
Nordic_Seas

NorthAtlantic = <earthdiagnostics.constants.Basin object>
North Atlantic Ocean

NorthPacific = <earthdiagnostics.constants.Basin object>
North Pacific Ocean

NorthWest_Passage = <earthdiagnostics.constants.Basin object>
NorthWest_Passage

North_Atlantic_Arctic = <earthdiagnostics.constants.Basin object>
North_Atlantic_Arctic

North_Hemisphere_Ocean = <earthdiagnostics.constants.Basin object>
North_Hemisphere_Ocean

Norwegian_Sea = <earthdiagnostics.constants.Basin object>
Norwegian_Sea

Okhotsk = <earthdiagnostics.constants.Basin object>
Okhotsk

OpenOcean = <earthdiagnostics.constants.Basin object>
OpenOcean

Pacific = <earthdiagnostics.constants.Basin object>
Pacific Ocean

Ross_Sea = <earthdiagnostics.constants.Basin object>
Ross_Sea

Serreze_Arctic = <earthdiagnostics.constants.Basin object>
Serreze_Arctic

Southern_Hemisphere = <earthdiagnostics.constants.Basin object>
Southern_Hemisphere

StLawr = <earthdiagnostics.constants.Basin object>
StLawr

Subpolar_Gyre = <earthdiagnostics.constants.Basin object>
Subpolar_Gyre

TotalArc = <earthdiagnostics.constants.Basin object>
TotalArc

TropicalAtlantic = <earthdiagnostics.constants.Basin object>
Tropical Atlantic Ocean

TropicalIndian = <earthdiagnostics.constants.Basin object>
Tropical Indian Ocean

TropicalPacific = <earthdiagnostics.constants.Basin object>
Tropical Pacific Ocean

Vilkitsky_Strait = <earthdiagnostics.constants.Basin object>
Vilkitsky_Strait

Weddell_Sea = <earthdiagnostics.constants.Basin object>
Weddell_Sea

Western_Central_Arctic = <earthdiagnostics.constants.Basin object>
Western_Central_Arctic

classmethod 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 name or 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

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

7.1.7 earthdiagnostics.datamanager

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

Class to manage the data repositories

Parameters **config** (Config) –

file_exists(*domain*, *var*, *startdate*, *member*, *chunk*, *grid=None*, *box=None*, *frequency=None*, *vartype=1*)
Checks if a given file exists

Parameters

- **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)

- **box** (*Box*) – file's box (only needed to retrieve sections or averages)
- **frequency** (*Frequency*) – 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

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

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

Parameters

- **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)
- **box** (*Box*) – file's box (only needed to retrieve sections or averages)
- **frequency** (*Frequency*) – 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

get_year(*domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*)

Ge a file containing all the data for one year for one variable :param domain: variable's domain :type domain: Domain :param var: variable's name :type var: str :param startdate: startdate to retrieve :type startdate: str :param member: member to retrieve :type member: int :param year: year to retrieve :type year: int :param grid: variable's grid :type grid: str :param box: variable's box :type box: Box :return:

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

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

Parameters

- **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)
- **box** (Box) – file's box (only needed to retrieve sections or averages)
- **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()
Prepares the data to be used by the diagnostic. :return:

send_file(*filetosend*, *domain*, *var*, *startdate*, *member*, *chunk=None*, *grid=None*, *region=None*, *box=None*, *rename_var=None*, *frequency=None*, *year=None*, *date_str=None*, *move_old=False*, *diagnostic=None*, *cmorized=False*, *vartype=1*)
Copies a given file to the CMOR repository. It also automatically converts to netCDF 4 if needed and can merge with already existing ones as needed

Parameters

- **move_old** (*bool*) – if true, moves files following older conventions that may be found on the links folder
- **date_str** – exact date_str to use in the cmorized file
- **year** (*int*) – if frequency is yearly, this parameter is used to give the corresponding year
- **rename_var** (*str*) – if exists, the given variable will be renamed to the one given by var
- **filetosend** (*str*) – path to the file to send to the CMOR repository
- **region** (*str*) – specifies the region represented by the file. If it is defined, the data will be appended to the CMOR repository as a new region in the file or will overwrite if region was already present
- **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)
- **box** (Box) – file's box (only needed to retrieve sections or averages)
- **frequency** (*Frequency*) – file's frequency (only needed if it is different from the default)
- **diagnostic** (Diagnostic) – diagnostic used to generate the file
- **cmorized** (*bool*) – flag to indicate if file was generated in cmorization process
- **vartype** (*VariableType*) – Variable type (mean, statistic)

Type str

class earthdiagnostics.datamanager.**NetCDFFile**(*remote_file*, *local_file*, *domain*, *var*, *cmor_var*, *data_convention*, *region*)
Bases: object

Class to manage netCDF file and pr

Parameters

- **remote_file** (*str*) –
- **local_file** (*str*) –
- **domain** (*Domain*) –
- **var** (*str*) –
- **cmor_var** (Variable) –

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

Class to manage unit conversions

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

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

classmethod get_conversion_factor_offset(*input_units*, *output_units*)
Gets 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

7.1.8 earthdiagnostics.diagnostic

class earthdiagnostics.diagnostic.**Diagnostic**(*data_manager*)
Bases: object

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

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

compute()
Calculates the diagnostic and stores the output

Must be implemented by derived classes

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*) – diagnostics manager
- **options** (*list[str]*) – list of strings containing the options passed to the diagnostic

Returns

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

Parameters **name** (*str*) – diagnostic alias

Returns the selected Diagnostic class, None if name can not be found

Return type *Diagnostic*

static register()
Register a new diagnostic using the given alias. It must be call using the derived class. :param cls: diagnostic class to register :type cls: Diagnostic

send_file(*filetosend*, *domain*, *var*, *startdate*, *member*, *chunk=None*, *grid=None*, *region=None*, *box=None*, *rename_var=None*, *frequency=None*, *year=None*, *date_str=None*, *move_old=False*, *vartype=1*)

Parameters

- **filetosend** –
- **domain** (*ModelingRealm*) –
- **var** –
- **startdate** –
- **member** –
- **chunk** –
- **grid** –
- **region** –
- **box** –
- **rename_var** –
- **frequency** (*Frequency*) –
- **year** –
- **date_str** –
- **move_old** –
- **vartype** (*VariableType*) – Variable type (mean, statistic)

Returns

7.1.9 earthdiagnostics.earthdiags

class earthdiagnostics.earthdiags.**EarthDiags**(*config_file*)
Bases: object

Launcher class for the diagnostics

Parameters **config_file** (*str*) – path to the configuration file

static parse_args()
Entry point for the Earth Diagnostics. For more detailed documentation, use -h option

run()
Run the diagnostics

7.1.10 earthdiagnostics.parser

class earthdiagnostics.parser.**Parser**(*defaults=None*, *dict_type=<class 'collections.OrderedDict'>*, *allow_no_value=False*)
Bases: ConfigParser.SafeConfigParser

Class to manage the config file. It add options to manage default values and to convert strings to the desired types (int, bool, list ...)

add_section(*section*)
Create a new section in the configuration.

Raise DuplicateSectionError if a section by the specified name already exists. Raise ValueError if name is DEFAULT or any of it's case-insensitive variants.

check_exists(*section*, *option*)
Checks if an option exists

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check

Returns True if option exists, False otherwise

Return type bool

check_is_boolean(*section*, *option*, *must_exist*)
Checks if an option is a boolean value

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist

Returns True if option value is boolean, False otherwise

Return type bool

check_is_choice(*section*, *option*, *must_exist*, *choices*)
Checks if an option is a valid choice in given self

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist
- **choices** (*list*) – valid choices

Returns True if option value is a valid choice, False otherwise

Return type bool

check_is_int (*section*, *option*, *must_exist*)
Checks if an option is an integer value

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist

Returns True if option value is integer, False otherwise

Return type bool

check_regex (*section*, *option*, *must_exist*, *regex*)
Checks if an option complies with a regular expression

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist
- **regex** (*str*) – regular expression to check

Returns True if option complies with regex, False otherwise

Return type bool

get (*section*, *option*, *raw=False*, *vars=None*)
Get an option value for a given section.

If ‘vars’ is provided, it must be a dictionary. The option is looked up in ‘vars’ (if provided), ‘section’, and in ‘defaults’ in that order.

All % interpolations are expanded in the return values, unless the optional argument ‘raw’ is true. Values for interpolation keys are looked up in the same manner as the option.

The section DEFAULT is special.

get_bool_option (*section*, *option*, *default=True*)
Gets a boolean option

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to get
- **default** (*bool*) – value to be returned if option is not present

Returns option value

Return type bool

get_float_option (*section*, *option*, *default=0.0*)
Gets a float option

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to get
- **default** (*float*) – value to be returned if option is not present

Returns option value

Return type float

get_int_option(*section*, *option*, *default=0*)
Gets an integer option

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to get
- **default** (*int*) – value to be returned if option is not present

Returns option value

Return type int

get_list_option(*section*, *option*, *default=[]*, *separator=' '*)
Gets a list option

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to get
- **default** (*object*) – value to be returned if option is not present
- **separator** (*str*) – separator used to split the list

Returns option value

Return type list

get_option(*section*, *option*, *default=''*)
Gets an option

Parameters

- **section** (*str*) – section that contains the option
- **option** (*str*) – option to get
- **default** (*object*) – value to be returned if option is not present

Returns option value

Return type str

has_option(*section*, *option*)
Check for the existence of a given option in a given section.

has_section(*section*)
Indicate whether the named section is present in the configuration.

The DEFAULT section is not acknowledged.

items(*section*, *raw=False*, *vars=None*)
Return a list of tuples with (name, value) for each option in the section.

All % interpolations are expanded in the return values, based on the defaults passed into the constructor, unless the optional argument 'raw' is true. Additional substitutions may be provided using the 'vars' argument, which must be a dictionary whose contents overrides any pre-existing defaults.

The section DEFAULT is special.

options (*section*)
Return a list of option names for the given section name.

read (*filenames*)
Read and parse a filename or a list of filenames.

Files that cannot be opened are silently ignored; this is designed so that you can specify a list of potential configuration file locations (e.g. current directory, user's home directory, systemwide directory), and all existing configuration files in the list will be read. A single filename may also be given.

Return list of successfully read files.

readfp (*fp*, *filename=None*)
Like read() but the argument must be a file-like object.

The 'fp' argument must have a 'readline' method. Optional second argument is the 'filename', which if not given, is taken from fp.name. If fp has no 'name' attribute, '<???>' is used.

remove_option (*section*, *option*)
Remove an option.

remove_section (*section*)
Remove a file section.

sections ()
Return a list of section names, excluding [DEFAULT]

set (*section*, *option*, *value=None*)
Set an option. Extend ConfigParser.set: check for string values.

write (*fp*)
Write an .ini-format representation of the configuration state.

7.1.11 earthdiagnostics.utils

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 ()
Removes all temporary files created with Tempfile until now

files = []
List of files to clean automatically

static get (*filename=None*, *clean=None*, *suffix='.nc'*)
Gets 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 ExecutionError
Bases: exceptions.Exception

Exception to raise when a command execution fails

exception Utils.**UnzipException**
Bases: exceptions.Exception

Excpetion raised when unzip fails

static Utils.**available_cpu_count**()
Number of available virtual or physical CPUs on this systemx

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

static Utils.**concat_variables**(*source*, *destiny*, *remove_source=False*)
Add variables from a nc file to another :param source: path to source file :type source: str :param destiny: path to destiny file :type destiny: str :param remove_source: if True, removes source file :type remove_source: bool

static Utils.**convert2netcdf4**(*filetoconvert*)
Checks if a file is in netCDF4 format and converts to netCDF4 if not

Parameters **filetoconvert** (*str*) – file to convert

static Utils.**copy_dimension**(*source*, *destiny*, *dimension*, *must_exist=True*, *new_names=None*)
Copies the given dimension from source to destiny, including dimension variables if present

Parameters

- **new_names** (*dict*) – dictionary containing variables to rename and new name as key-value pairs
- **source** (*netCDF4.Dataset*) – origin file
- **destiny** (*netCDF4.Dataset*) – destiny file
- **dimension** (*str*) – variable to copy
- **must_exist** (*bool*) – if false, does not raise an error uf variable does not exist

Returns

static Utils.**copy_file**(*source*, *destiny*)
Copies a file from source to destiny, creating dirs if necessary

Parameters

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

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

Parameters

- **add_dimensions** (*bool*) – if it's true, dimensions required by the variable will be automatically added to the file. It will also add the dimension variable
- **source** (*netCDF4.Dataset*) – origin file
- **destiny** (*netCDF4.Dataset*) – destiny file
- **variable** (*str*) – variable to copy
- **must_exist** (*bool*) – if false, does not raise an error uf variable does not exist
- **new_names** (*dict*) – dictionary containing variables to rename and new name as key-value pairs

Returns

static Utils.**create_folder_tree**(*path*)
Createas a fodle path will and parent directories if needed. :param path: folder's path :type path: str

static Utils.**execute_shell_command**(*command*, *log_level=10*)
Executes a sheel command :param command: command to execute

Log.info('Detailed time for diagnostic class') :param log_level: log level to use for command output :type log_level: int :return: command output :rtype: list

static Utils.**expand_path**(*path*)
Expands character ~ and system variables on the given path :param path: path to expand :type path: str :return: path after the expansion

static Utils.**get_datetime_from_netcdf**(*handler*, *time_variable='time'*)
Gets a datetime array from a netCDF file

Parameters

- **handler** (*netCDF4.Dataset*) – file to read
- **time_variable** (*str*) – variable to read, by default 'time'

Returns Datetime numpy array created from the values stored at the netCDF file

Return type np.array

static Utils.**get_file_hash**(*filepath*)
Returns the MD5 hash for the given filepath :param filepath: path to the file to compute hash on :type filepath:str :return: file's MD5 hash :rtype: str

static Utils.**get_mask**(*basin*)
Returns a numpy array containing the mask for the given basin

Parameters **basin** (Basin) – basin to retrieve

Returns mask

Return type numpy.array

static Utils.**move_file**(*source*, *destiny*)
Moves a file from source to destiny, creating dirs if necessary

Parameters

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

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

static Utils.**openCdf**(*filepath*, *mode='a'*)
Opens a netCDF file and returns a handler to it

Parameters

- **filepath** (*str*) – path to the file
- **mode** (*str*) – mode to open the file. By default, a (append)

Returns handler to the file

Return type netCDF4.Dataset

static Utils.**remove_file**(*path*)
Removes a file, checking before if its exists

Parameters **path** (*str*) – path to file

static Utils.**rename_variable**(*filepath*, *old_name*, *new_name*, *must_exist=True*, *rename_dimension=False*)
Rename multiple variables from a NetCDF file :param filepath: path to file :type filepath: str :param old_name: variable's name to change :type old_name: str :param new_name: new name :type new_name: str :param must_exist: if True, the function will raise an exception if the variable name does not exist :type must_exist: bool :param rename_dimension: if True, also rename dimensions with the same name :type rename_dimension: bool

static Utils.**rename_variables**(*filepath*, *dic_names*, *must_exist=True*, *rename_dimension=False*)
Rename multiple variables from a NetCDF file :param filepath: path to file :type filepath: str :param dic_names: dictionary containing old names as keys and new names as values :type dic_names: dict :param must_exist: if True, the function will raise an exception if the variable name does not exist :type must_exist: bool :param rename_dimension: if True, also rename dimensions with the same name :type rename_dimension: bool

static Utils.**setminmax**(*filename*, *variable_list*)
Sets the valid_max and valid_min values to the current max and min values on the file :param filename: path to file :type filename: str :param variable_list: list of variables in which valid_min and valid_max will be set :type variable_list: str | list

static Utils.**untar**(*files*, *destiny_path*)
Untar files to a given destiny :param files: files to unzip :type files: list[Any] | Tuple[Any] :param destiny_path: path to destination folder :type destiny_path: str

static Utils.**unzip**(*files*, *force=False*)
Unzip a list of files :param files: files to unzip :type files: list :param force: if True, it will overwrite unzipped files :type force: bool

7.1.12 earthdiagnostics.variable

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

Class to characterize a CMOR variable. It also contains the static method to make the match between thje original name and the standard name. Requires data _convetion to be available in cmor_tables to work.

class earthdiagnostics.variable.**VariableAlias**(*alias*)
Bases: object

Class to characterize a CMOR variable. It also contains the static method to make the match between thje original name and the standard name. Requires data _convetion to be available in cmor_tables to work.

7.2 earthdiagnostics.general

7.2.1 earthdiagnostics.general.attribute

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

Bases: *earthdiagnostics.diagnostic.Diagnostic*

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 = 'att'
Diagnostic alias for the configuration file

compute()
Runs the diagnostic

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

Parameters

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

Returns

7.2.2 earthdiagnostics.general.monthlymean

class earthdiagnostics.general.monthlymean.**MonthlyMean**(*data_manager*, *startdate*, *member*, *chunk*, *domain*, *variable*, *frequency*, *grid*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Calculates monthly mean for a given variable

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
- **frequency** (*str*) – original frequency
- **grid** (*str*) – original data grid

alias = 'monmean'
Diagnostic alias for the configuration file

compute()
Runs the diagnostic

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

Parameters

- **diags** (*Diags*) – Diagnostics manager class
- **options** (*list[str]*) – variable, domain, frequency=day, grid='‘

Returns

7.2.3 earthdiagnostics.general.relink

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

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

Parameters

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

Returns

7.2.4 earthdiagnostics.general.rewrite

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

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()
Runs the diagnostic

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

Parameters

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

Returns

7.2.5 earthdiagnostics.general.scale

class earthdiagnostics.general.scale.**Scale**(*data_manager*, *startdate*, *member*, *chunk*, *value*, *offset*, *domain*, *variable*, *grid*, *min_limit*, *max_limit*)
Bases: *earthdiagnostics.diagnostic.Diagnostic*

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()
Runs the diagnostic

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

Parameters

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

Returns

7.3 earthdiagnostics.ocean

7.3.1 earthdiagnostics.ocean.areamoc

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

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

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()
Runs the diagnostic

classmethod generate_jobs(*diags*, *options*)
Creates 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

7.3.2 earthdiagnostics.ocean.averagesection

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

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()
Runs the diagnostic

classmethod generate_jobs(*diags*, *options*)
Creates 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

7.3.3 earthdiagnostics.ocean.convectionsites

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()
Runs the diagnostic

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

Parameters

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

Returns

7.3.4 earthdiagnostics.ocean.cutsection

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()
Runs the diagnostic

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

Parameters

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

Returns

7.3.5 earthdiagnostics.ocean.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()
Runs the diagnostic

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

Parameters

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

Returns

7.3.6 earthdiagnostics.ocean.heatcontent

class earthdiagnostics.ocean.heatcontent.**HeatContent**(*data_manager*, *startdate*, *member*, *chunk*, *basin*, *mixed_layer*, *box*)
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()
Runs the diagnostic

classmethod generate_jobs(*diags*, *options*)
Creates 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

7.3.7 earthdiagnostics.ocean.heatcontentlayer

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()
Runs the diagnostic

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

Parameters

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

7.3.8 earthdiagnostics.ocean.interpolate

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

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()
Runs the diagnostic

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

Parameters

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

Returns

7.3.9 earthdiagnostics.ocean.interpolatecdo

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

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()
Runs the diagnostic

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

Parameters

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

Returns

7.3.10 earthdiagnostics.ocean.maxmoc

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

Compute an Atlantic MOC index by finding 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()
Runs the diagnostic

classmethod generate_jobs(*diags*, *options*)
Creates 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

7.3.11 earthdiagnostics.ocean.mixedlayerheatcontent

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()
Runs the diagnostic

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

Parameters

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

Returns

7.3.12 earthdiagnostics.ocean.mixedlayersaltcontent

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()
Runs the diagnostic

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

Parameters

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

Returns

7.3.13 earthdiagnostics.ocean.moc

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()
Runs the diagnostic

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

Parameters

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

Returns

7.3.14 earthdiagnostics.ocean.psi

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()
Runs the diagnostic

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

Parameters

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

Returns

7.3.15 earthdiagnostics.ocean.siasiesiv

class earthdiagnostics.ocean.siasiesiv.**Siasiesiv**(*data_manager*, *startdate*, *member*, *chunk*, *basin*, *mask*)
Bases: *earthdiagnostics.diagnostic.Diagnostic*

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

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

Contributor Neven Fuckar <neven.fuckar@bsc.es>

Contributor Ruben Cruz <ruben.cruzgarcia@bsc.es>

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

Created April 2012

Last modified June 2016

alias = 'siasiesiv'
Diagnostic alias for the configuration file

compute()
Runs the diagnostic

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

Parameters

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

Returns

7.3.16 earthdiagnostics.ocean.verticalmean

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()
Runs the diagnostic

classmethod generate_jobs(*diags*, *options*)
Creates 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

7.3.17 earthdiagnostics.ocean.verticalmeanmeters

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

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()
Runs the diagnostic

classmethod generate_jobs(*diags*, *options*)
Creates 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

7.4 earthdiagnostics.statistics

7.4.1 earthdiagnostics.statistics.climatologicalpercentile

class earthdiagnostics.statistics.climatologicalpercentile.**ClimatologicalPercentile**(*data_manager*, *domain*, *variable*, *leadtimes*, *num_bins*, *experiment_config*)

Bases: *earthdiagnostics.diagnostic.Diagnostic*

Calculates the climatological percentiles for the given leadtimes

Parameters

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

alias = 'climpercent'
Diagnostic alias for the configuration file

compute()
Runs the diagnostic

classmethod generate_jobs(*diags*, *options*)
Creates 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

7.4.2 earthdiagnostics.statistics.monthlypercentile

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

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()
Runs the diagnostic

classmethod generate_jobs(*diags*, *options*)
Creates 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

e

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W