Earth Diagnostics Documentation

Release 3.0.0b

BSC-CNS Earth Sciences Department

Jul 06, 2016

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 and you will be mentioned here.

1.1 Installation

For now, you only have an option: dowload 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
- Acces to CDFTOOLS_3.0 executables for BSC-ES. At this point, those are located at /home/Earth/jvegas/CDFTOOLS_CMOR/bin.

Call the diags with -h option to see if everything is ready:

```
${PATH_TO_REPOSITORY}/earthdiagnostics.diags.py -h
```

1.2 Creating a config file

If you go into the earthdiagnsotics folder in the git repository, you will see a diags.conf that can be used as a model for your config file. It contains commentaries explaining what represents each one of its parameters, so please read it carefully.

Once you have configured your experiment you can execute any diagnostic by calling this command (substitute the variables for the real paths before launching, please)

```
${PATH_TO_REPOSITORY}/earthdiagnostics.diags.py ${PATH_TO_MY_CONF_FILE}
```

And... that's it. You will find your results inside CMOR's folder tree and a folder for the temp files in the scratch. This folder is named after the EXPID.

CHAPTER

TWO

TIPS AND TRICKS

2.1 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 time consumption is low enough to allow you keep using the machine while running, specially if you reserve a pair of cores for other uses.

2.2 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.

2.3 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

THREE

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: Reading is good. 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.

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.

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.

CHAPTER

FOUR

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.

4.1 Developing a diagnostic

For new diagnsotics 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

FIVE

FREQUENTLY ASKED QUESTIONS

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

CHAPTER

SIX

MODULE DOCUMENTATION

6.1 earthdiagnostics

6.1.1 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

6.1.2 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

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

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

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

6.1.4 earthdiagnostics.datamanager

class earthdiagnostics.datamanager.**DataManager**(*exp_manager*, *institution*, *model*, *expid*, *datafolder*, *frequency*, *experiment_name*, *scratch_dir*, *nfrp*, *calendar='standard'*)
Bases: object

Class to manage the data repositories

Parameters

- **exp_manager** (ExperimentManager) –
- **institution** (*str*) – CMOR's institution name
- **model** (*str*) – CMOR's model name
- **expid** (*str*) – experiment identifier
- **datafolder** (*str*) – Folder containing the data
- **frequency** (*str*) – default frequency
- **experiment_name** (*str*) – CMOR's experiment name

- **scratch_dir** (*str*) – path to scratch folder
- **nfrp** (*int*) – sample frequency
- **calendar** (*str*) – experiment's calendar

static domain_abbreviation(*domain*, *frequency*)
Returns the table name for a domain-frequency pair :param domain: variable's domain :type domain: str :param frequency: variable's frequency :type frequency: str :return: variable's table name :rtype: str

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

Parameters

- **file_path** (*str*) – path to the file
- **handler** (*netCDF$.Dataset*) – netCDF4 handler for the file
- **frequency** (*str*) – variable's frequency
- **member** (*int*) – member
- **startdate** (*str*) – startdate
- **temp** (*str*) – temporal file to use
- **variable** (*str*) – variable's name

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

Parameters

- **domain** (*str*) – 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)

Returns path to the copy created on the scratch folder

Return type str

get_year(*domain*, *var*, *startdate*, *member*, *year*, *grid=None*, *box=None*)
Gets all the data corresponding to a given year from the CMOR repository to the scratch folder as one file and returns the path to the scratch's copy.

Parameters

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

- **member** (*int*) – file's member
- **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)

Returns path to the copy created on the scratch folder

Return type str

prepare_CMOR_files(*force_rebuild*, *ocean*, *atmosphere*)

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.

Parameters

- **atmosphere** (*bool*) – activates atmosphere files cmorization
- **ocean** (*bool*) – activates ocean files cmorization
- **force_rebuild** (*bool*) – if True, forces the creation of the CMOR files

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*)

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

- **date_str** –
- **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** (*str*) – 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)

Returns path to the copy created on the scratch folder

Return type str

class earthdiagnostics.datamanager.**Variable**(*line*)
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 cmor_table.csv to work.

classmethod get_variable(*original_name*)
Returns the cmor variable instance given a varuiable name :param original_name: original variable's name :type original_name: str :return: CMOR variable :rtype: Variable

6.1.5 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

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(*alias*)
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 :param alias: alias for the diagnostic :type alias: str

6.1.6 earthdiagnostics.diags

class earthdiagnostics.diags.**Diags**(*config_file*)
Bases: object

Launcher class for the diagnostics

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

run()
Run the diagnostics

earthdiagnostics.diags.**main**()
Entry point for the Earth Diagnostics. For more detailed documentation, use -h option

6.1.7 earthdiagnostics.experimentmanager

class earthdiagnostics.experimentmanager.**ExperimentManager**(*startdates*, *members*, *num_chunks*, *chunk_size*, *member_digits*, *calendar='standard'*)

Bases: object

Encapsulates all chunk related tasks

Parameters

- **startdates** (*list[str]*) – startdates to run
- **members** (*list[int]*) – members to run
- **num_chunks** (*int*) – chunks to run
- **chunk_size** (*int*) – length of the chunks in months
- **member_digits** (*int*) – minimum number of digits to use in the member name
- **calendar** (*str*) – experiment's calendar

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]

6.1.8 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

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

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

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

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

static copy_variable(*source*, *destiny*, *variable*, *must_exist=True*)
Copies the given variable from source to destiny

Parameters

- **source** (*netCDF4.Dataset*) – origin file
- **destiny** (*netCDF4.Dataset*) – destiny file
- **variable** (*str*) – variable to copy
- **must_exist** (*boo°*) – if false, does not raise an error uf variable does not exist

Returns

static 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 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 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 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

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

static 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 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 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 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

6.2 earthdiagnostics.ocean

6.2.1 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

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

6.2.2 earthdiagnostics.ocean.averagesection

class earthdiagnostics.ocean.averagesection.**AverageSection**(*data_manager*, *startdate*, *member*, *chunk*, *variable*, *domain*, *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** (*str*) – variable's domain
- **box** (Box) – box to use for the average

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, minimun longitude, maximun longitude, minimum latitude, maximum latitude, domain=ocean

Returns

6.2.3 earthdiagnostics.ocean.convectionsites

class earthdiagnostics.ocean.convectionsites.**ConvectionSites**(*data_manager*, *startdate*, *member*, *chunk*, *nemo_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
- **nemo_version** (*str*) – NEMO's version

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

6.2.4 earthdiagnostics.ocean.cutsection

class earthdiagnostics.ocean.cutsection.**CutSection**(*data_manager*, *startdate*, *member*, *chunk*, *variable*, *domain*, *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** (*str*) – variable's domain
- **zonal** (*bool*) – specifies if section is zonal or meridional
- **value** (*int*) – value of the section's coordinate

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

6.2.5 earthdiagnostics.ocean.gyres

class earthdiagnostics.ocean.gyres.**Gyres**(*data_manager*, *startdate*, *member*, *chunk*, *nemo_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
- **nemo_version** (*str*) – NEMO's version

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

6.2.6 earthdiagnostics.ocean.interpolate

class earthdiagnostics.ocean.interpolate.**Interpolate**(*data_manager*, *startdate*, *member*, *chunk*, *variable*, *domain*, *nemo_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 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** (*str*) – variable's domain
- **nemo_version** (*str*) – NEMO model version

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=ocean

Returns

6.2.7 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

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

6.2.8 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

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

6.2.9 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

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

6.2.10 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

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

6.2.11 earthdiagnostics.ocean.siasiesiv

class earthdiagnostics.ocean.siasiesiv.**Siasiesiv**(*data_manager*, *basin*, *startdate*, *member*, *chunk*, *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

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

6.2.12 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

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

6.2.13 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

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

e

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W