Supplement of Geosci. Model Dev. Discuss., 8, 7541–7661, 2015
http://www.geosci-model-dev-discuss.net/8/7541/2015/
doi:10.5194/gmdd-8-7541-2015-supplement

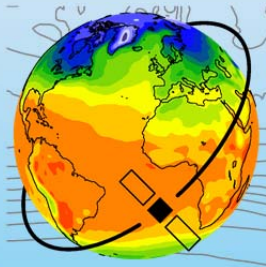
Geoscientific Model Development Discussions

Supplement of

ESMValTool (v1.0) – a community diagnostic and performance metrics tool for routine evaluation of Earth System Models in CMIP

V. Eyring et al.

Correspondence to: V. Eyring (veronika.eyring@dlr.de)



ESMValTool

Earth System Model eValuation Tool

Version 1.0 (v1.0)

User's and Developer's Guide

July 2015

http://www.pa.op.dlr.de/ESMValTool/

Contacts: Veronika.Eyring@dlr.de and Axel.Lauer@dlr.de

ESMValTool v1.0 user's and developer's guide

Contents

Preface

This user's and developer's guide consists of parts targeting two overlapping categories of scientists working with the Earth System Model Evaluation Tool (ESMValTool):

(1) Part I: User's Guide: this part gives an introduction to the ESMValTool (v1.0) including installation, running the ESMValTool, and available user settings of existing diagnostics and performance metrics. The target group would typically consist of scientists mostly interested in running the ESMValTool as provided either on CMIP model simulations or on mode simulations performed with other models, and on observations.

(2) Part II: Developer's Guide: this part gives additional technical details on the ESMValTool not necessarily needed to apply the ESMValTool as well as an introduction to implementing new variables and new diagnostics. This part is mostly intended for scientists interested in technical details as well as in contributing to the development of the ESMValTool by adding new nameslists and code for additional diagnostics or performance metrics.

For the developer's guide (part II), it is assumed that the user/developer is already familiar with the ESMValTool user guide introduced in part I.

Part I:User's Guide

1 Introduction

The Earth System Model Evaluation Tool (ESMValTool) is a community-development that aims at improving diagnosing and understanding of the causes and effects of model errors and inter-model spread. The ESMValTool is open to both users and developers encouraging open exchange of diagnostic source code and evaluation results from the Coupled Model Intercomparison Project (CMIP) ensemble. This will facilitate and improve ESM evaluation beyond the state-of-the-art and aims at supporting the activities within CMIP and at individual modelling centers. We envisage running the ESMValTool routinely on the CMIP model output utilizing observations available through the Earth System Grid Federation (ESGF) in standard formats (obs4MIPs) or made available at ESGF nodes.

The goal is to develop a benchmarking and evaluation tool that produces well-established analyses as soon as model output from CMIP simulations becomes available, e.g., at one of the central repositories of the ESGF. This is realized through standard namelists that reproduce a certain set of diagnostics and performance metrics that have demonstrated its importance in benchmarking Earth System Models (ESMs) in a paper or assessment report, such as Chapter 9 of the Intergovernmental Panel on Climate Change (IPCC) Fifth Assessment Report (AR5) (Flato et al., 2013). The expectation is that in this way a routine and systematic evaluation of model results can be made more efficient, thereby enabling scientists to focus on developing more innovative methods of analysis rather than constantly having to "reinvent the wheel".

In parallel to standardization of model output, the ESGF also hosts observations for Model Intercomparison Projects (obs4MIPs). Obs4MIPs provides open access data sets of satellite data that are comparable in terms of variables, temporal and spatial frequency, and periods to the Coupled Model Intercomparison Project (CMIP) data set (Taylor et al., 2012). The ESMValTool utilizes these observations plus additionally available observations in order to evaluate the models performance. In many diagnostics and metrics, more than one observational data set or meteorological reanalysis is used to assess uncertainties in observations.

1.1 Objectives and Approach

The main idea of the ESMValTool is to provide a broad suite of diagnostics which can be performed easily when new model simulations are run. The suite of diagnostics needs to be broad enough to reflect the diversity and complexity of Earth System Models, but must also be robust enough to be run routinely or semi-operationally.

In order the address these challenging objectives the ESMValTool is conceived as a framework which allows community contributions to be bound into a coherent framework.

1.2 Architecture

Figure 1 shows a schematic of the ESMValTool architecture: the workflow manager (controlled by the Python script "main.py") runs a set of diagnostics on data provided by, for instance, a data archive. The configuration and the settings of each diagnostic are specified in namelists read and passed to the diagnostics by the workflow manager. The results which typically comprise of NetCDF files and/or plots are stored in output folders along with log-files summarizing the data used, references, and technical details to ensure traceability and reproducibility of the results.

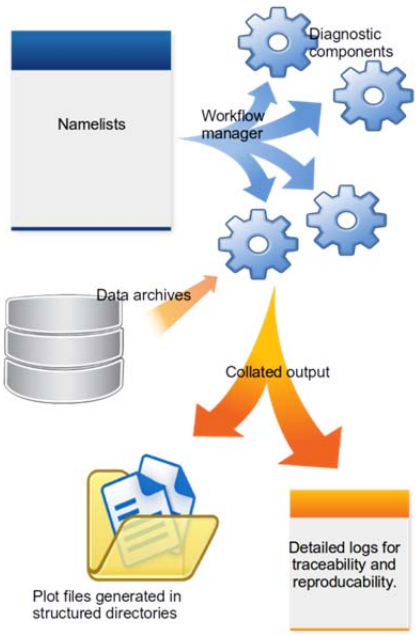


Figure 1 Schematic of the system architecture. The workflow manager (main.py) passes information to the diagnostics; results and log-files are written to dedicated folders.

2 Software installation

2.1 Prerequisites

The ESMValTool has the following software requirements (note that specific diagnostics might require additional software packages):

1. Unix(-like) operating system
2. Python version 2.7.x for running the Python script *main.py*;
3. NCAR Command Language (NCL 2014) version 6.2 or higher to run the quality check and reformat routines processing all input files. See the control flow description on *reformat_default* in Table S10 for details.
4. Input files in netCDF with required attributes and dimension names. Valid input files are:
    1. a CMIP or similarly standardized format using a CMIP5 table, or with discrepancies that can be handled via the definitions in the files *reformat_scripts/recognized_units.dat* and *reformat_scripts/recognized_units.dat*, respectively.
    2. any input file with a (user-)supplied reformat routine that converts the input data during run-time, see the control flow description on *reformat_EMAC* in Table S10 for details

- Common GNU utilities such as "*wc*", "*date*", "*basename*", and "*more*", which are usually part of the standard Linux distribution

2.2 Obtaining the source code

The ESMValTool will be made available at http://www.pa.op.dlr.de/ESMValTool via a tar-file with a doi assigned. The ESMValTool will be released under the Apache License, version 2.0 and citation of the ESMValTool paper *Eyring et al. (2015), Geosci. Model Dev. Discuss.* is kindly requested upon use. In addition, ESMValTool will be further developed in a version controlled repository (see section 12.1) that is accessible only to the development team. The wider climate community is encouraged to contribute to this effort and to join the ESMValTool development team for contribution of additional more in-depth diagnostics for ESM evaluation. A wiki page (see section 12.3) that describes ongoing developments is also available. Interested users and developers are welcome to contact the core development team (see section 9).

2.3 Software installation

The tar-ball can be unpacked with the standard tar command, e.g.,

tar –xvf ESMValTool_v1.0.tar

3 ESMValTool namelists

The ESMValTool namelists are the “control centers” acting as interfaces between the user and the various scripts and configuration files that make up the ESMValTool. A namelist specifies a list of diagnostics to run, global flags and a list of models and observations that are used within the diagnostics. Namelists are text files written in XML (EXtensible Markup Language) [XML]. As a simple text file, the XML-namelist can be easily modified by the user.

For any given namelist “*namelist.xml”*, the ESMValTool is invoked from the command line via:

python main.py nml/namelist.xml

The Python “workflow manager” *main.py* will parse the namelist (*namelist.xml*) and call all diagnostic scripts listed in the namelist. This sequence is schematically depicted in Figure 2 and involves the following steps:

1. parse the namelist
2. identify the input files on the file system
3. run an NCL script to check and reformat the input files
4. if needed, run a NCL script to compute derived variables such as, for instance, climate indices
5. run the diagnostic script (NCL/Python/R/etc.)
6. repeat previous steps until all diagnostics listed in the namelist are processed

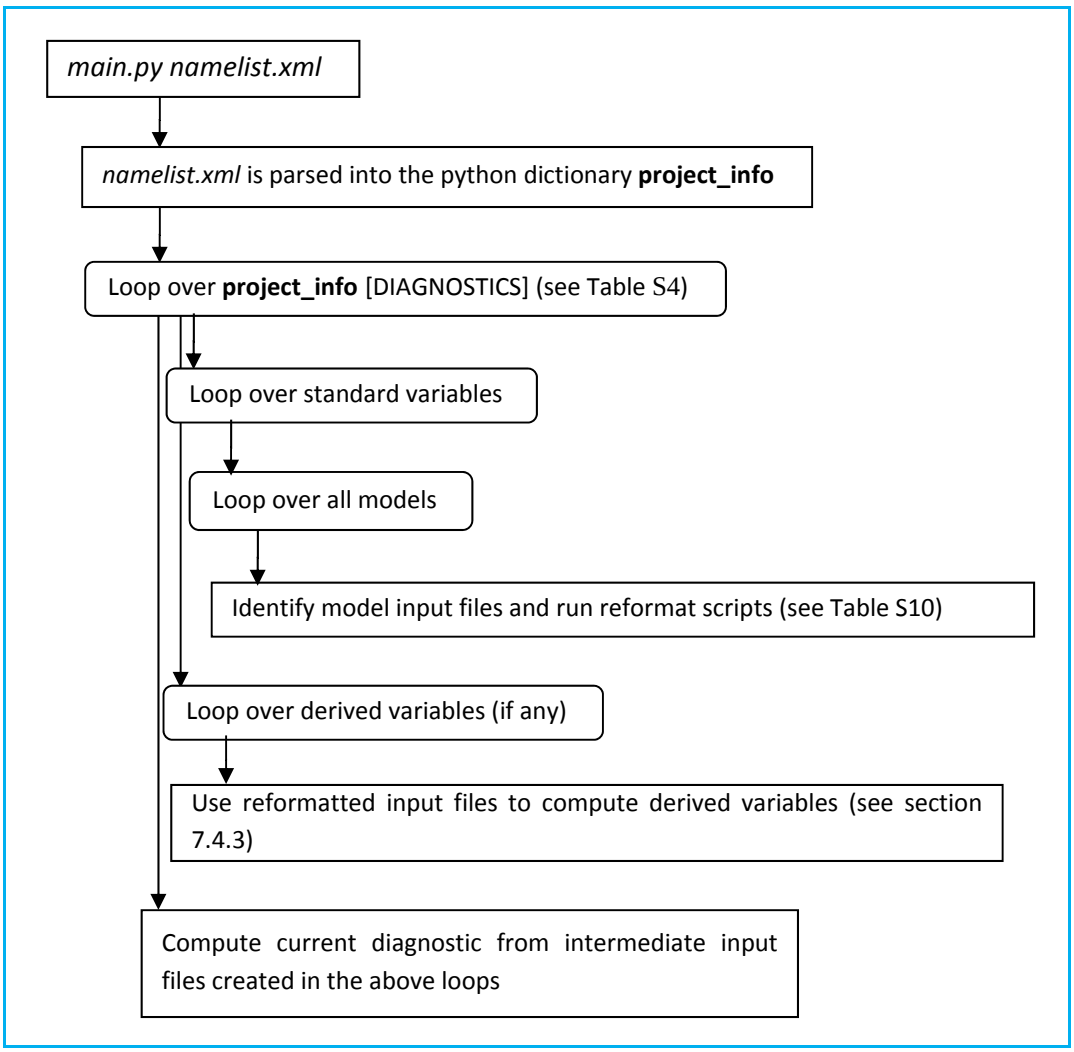


Figure 2 ESMValTool control flow.

The script *main.py* processes the information in the XML namelist to be used by each of the supported programming languages (currently NCL, Python and R) used for the diagnostic scripts. This means that different diagnostics, even if implemented in different programming languages, can be called within the same namelist. Any changes to the settings of the namelist will passed to each diagnostic script.

Note that the coupling between the namelist and the diagnostic scripts is "loose". The Python workflow manager *main.py* passes all information in the namelist to the target diagnostic script, e.g., via intermediate files or environment variables, but it is up to the diagnostic script to act on that information.

Basic structure of a namelist

<GLOBAL>

controls the general settings (see Table S1) ; see section 3.1, “More on the <GLOBALS>-tag” below for details

</GLOBAL>

<MODELS>

defines the models/observations and years to be processed and their pathnames; see the section 3.2, “More on the <MODELS>-tag” below for details

</MODELS>

<DIAGNOSTIC>

defines which diagnostics are run (see Table S4); each diagnostic is enclosed in an opening <diag> and closing </diag>-tag; see the section 3.3, “More on the <DIAGNOSTICS>-tag” below for details

</DIAGNOSTIC>

Please note that the “loose coupling” described above applies particularly to the settings defined in the two elements <GLOBAL> and <DIAGNOSTIC>.

3.1 More on the <GLOBAL>-tag

Table S1 summarizes the tags defined in the <GLOBAL> section of the namelist. Some of these tags (e.g., regridding_dir) are specific to some diagnostics and not generally defined in all namelists.

Table S1 Tags of the <GLOBAL> section of the namelist. Note that not all tags might be used by a diagnostic.

Name	Type	Description
write_plots	boolean	Produce plots
write_netcdf	boolean	Write results to netCDF file
force_processing	boolean	Force certain intermediate files (netCDF) to be recreated instead of using cached files
wrk_dir	string	Output path for data (netCDF, acknowledgements)
plot_dir	string	Output path for plots
climo_dir	string	Path for intermediate files (netCDF)
show_debuginfo	string	Generate a second version of each figure with explanatory text overlayed
regridding_dir	string	Path for intermediate files used for NCL regridding routines
max_data_filesize	integer	Limits internal memory handling in some core NCL scripts
verbosity	integer	Verbosity level (0 = minimum output, 4=maximum output)
exit_on_warning	boolean	Stop on warnings
output_file_type	string	File format of plots (ps, pdf, eps, png); not all formats supported by all diagnostic scripts

3.2 More on the <MODELS>-tag

Each data set is specified by a <model> line with the first entry of each model line being the “project specifier” (see Table S2). The project specifier refers to a Python class that is used to parse the model line. For example, a model line with the “CMIP5” specifier looks like:

<model> CMIP5 name mip experiment ensemble start-year end-year path </model>

The project specifier “CMIP5” will search for files in “path” with filenames matching the pattern

_mip_name_experiment_ensemble_

Here, the initial asterisk is a placeholder for the variable, which is defined in the <DIAGNOSTICS>-tag (see below), question marks are placeholders for the start/end months of the data set. This naming convention conforms to the syntax used for CMIP5 DRS filenames (as implied by the project specifier name). By implementing their own project specifier classes into the Python code (*interface_scripts/projects.py*), the user can handle data sets that follow different file naming conventions or require additional information to be passed along in addition to the filename. Table S2 gives a summary of the available project specifiers and arguments to be used in each <model> line.

The <model>-tag may also take the optional attribute "*id*":

<model id="string">

The attribute *id* specifies a string that can be used to refer to the model in other places of the namelist. Table S3 gives a summary of valid attributes in <model>-tags.

Table S2 Project specifiers and corresponding arguments.

project specifier	argument 1	argument 2	argument 3	argument 4	argument 5	argument 6	argument 7	argument 8
ana4mips	name	table	experiment	ensemble	realm	start year	end year	path
CCMVal	name	case	ensemble	start year	end year	path	-	-
CCMVal1	name	name	ensemble	start year	end year	path	-	-
CCMVal2	name	case name case name	ensemble	start year	end year	path	-	-
CMIP5	name	table	experiment	ensemble	start year	end year	Path	-
CMIP5_ETHZ	name	table	ensemble	start year	end year	path	-	
CMIP5_gridfile	name	table	experiment	ensemble	start year	end year	path	gridfile
CMIP5_SMHI	name	table	experiment experiment experiment	ensemble	start year	end year	frequency	path
ECEARTH	name	experiment	ensemble	start year	end year	path	-	-
EMAC	name	ensemble	start year	end year	path	-	-	-
MiKlip	name	table	experiment	ensemble	realm	start year	end year	path
MiKlip_baseline0	name	table	experiment	ensemble	realm	start year	end year	path
OBS	name	case name	ensemble	start year	end year	Path	-	-
OBS_gridfile	name	case name (insitu, sat, ground, reanaly)	ensemble	start year	end year	path	gridfile	-
obs4mips	name	process level	ensemble	start year	end year	path	-	-

Table S3 Optional attributes of the <model> tag.

Name	Type	Description
id	string	Define a name used to refer to the model data in other parts of the namelist

3.3 More on the <DIAGNOSTICS>-tag

Each <diag> entry refers to one or several scripts in the folder *diag_scripts/*, complemented by a variable name (see Table S7 for a list of variables; please note that the list of variables is constantly extended and check the ESMValTool wiki page (see section 12.3) for the most recent list) and the corresponding (input) field type (see Table S6). Optionally the <diag>-tag may contain additional <model>-tags; these data sets will be processed only by the diagnostic(s) listed in the current <diag> entry. In this way it is possible to define a set of models to be analyzed by all diagnostics in the namelist (in the <MODELS> section) and a set of models to be analyzed only by specific diagnostics (in the <diag> section). Available <diag>-tags are listed in Table S4, their optional attributes in Table S5.

Table S4 Tags of the <diag> section within the <DIAGNOSTICS> section of the namelist. There are no default values.

Name	Type	Description
description	string	1-line description / title of the diagnostic
variable_def_dir	string	Path for the variable-specific configuration file (usually *variable_defs*)
variable	string	Variable name: a script with the same name (*variable_defs/<variable>.ncl*) defines the variable to process (see Table S7 for a list of variables) including possible preprocessing (e.g., calculating derived variables). Variable scripts should be located in the local folder *variable_defs* and written in NCL. Even though the variable scripts are written in NCL all meta data defined in the scripts are passed on to the target diagnostic script regardless of the used language (via variable attributes). If multiple variables need to be passed on to a diagnostic script, multiple <variable>-tags have to be defined.
field_type	string	Type of input field (see Table S6) that can be used by the diagnostic scripts. If multiple <variable>-tags are defined a single (which is then applied to all) or an equal number of <field type>-tags has to be defined.
diag_script_cfg_dir	string	Path for diagnostic script configuration file
diag_script	string	Name of diagnostic script; the script can be written in any language currently supported by ESMValTool (NCL, R and Python) and has to be located in the local folder *diag_scripts*. The settings defined in the diagnostic script configuration file defined by the diag_script *cfg* attribute is loaded at the beginning of the diagnostic script.
model (optional)	string	Additional data sets specific for this <diag>-section. Data sets defined here will be processed in addition to the ones defined in the *MODELS* section (see above) but will be ignored by other <diag>-sections.

Table S5 Optional attributes of selected tags in the <diag> section.

Name	Type	Parent tag	Description
ref_model	string	<variable>	Defines this data set as the reference data set within the diagnostic. The string *ref_model* refers to either the model name, as specified in Table S2, or the model attribute *id* as specified in Table S3. Note that because both model and observational data sets are specified via the <model>-tag any of them can be used as a reference data set.
exclude	string	<variable>	When using more than one variable corresponding to different observational data sets (e.g., precipitation and skin temperature), it is necessary to use this attribute to match which variable goes with which data set, e.g., pr with TRMM and ts with HadISST using, <variable ref_model="trmm" exclude="hadisst"> pr … <variable ref_model="hadisst" exclude="trmm"> ts …
cfg	string	<diag_script>	Configuration file for the diagnostic script

Table S6 Field types.

Name	Description
T2Ms	Monthly-mean 2d atmosphere or land surface data (longitude, latitude, time:month)
T3M	Monthly-mean 3d atmosphere data (longitude, latitude, pressure, time:month)
T2Mz	Monthly-mean zonal mean 2d atmosphere or land surface data (longitude, pressure, time:month)

T1Ms	Monthly-mean 1d atmosphere or land surface data on a certain pressure level (latitude, time:month)
T2Ds	Daily-mean 2d atmosphere data (longitude, latitude, time:day)
T3D	Daily-mean 3d atmosphere data (longitude, latitude, pressure, time:day)
T2Dz	Daily-mean zonal mean 2d atmosphere data (latitude, pressure, time:month)
T2Is	Daily instantaneous 2d atmosphere data for all years (longitude, latitude, time:day)
T3I	Daily-instantaneous 3d atmosphere data for selected years (longitude, latitude, model level, time:day)
T2Iz	Daily instantaneous zonal mean 2d atmosphere data for all years (latitude, pressure, time:day)
T1Iz	Daily instantaneous 1d field for all years (latitude-pressure, time:day)
T0I	Daily instantaneous 0d field for all years (time:day)
T0As	Annual-mean 0d atmosphere or land surface data on a certain pressure level (latitude, time:year)
F2Ms	Constant 2d land surface data (latitude, longitude)
TO2Ms	Monthly-mean 2d ocean or sea ice data (longitude, latitude, time:month)
TO3M	Monthly-mean 3d ocean or sea ice data (longitude, latitude, model level, time:month)

Table S7 Variable definition scripts.

Script name	Description
clivi.ncl	Vertically integrated cloud ice
cl.ncl	Cloud area fraction (3d)
clt.ncl	Total cloud fraction
clwvi.ncl	Vertically integrated total cloud water (ice + liquid)
concbc.ncl	
concbcSTP.ncl	
conccnmode.ncl	
conccnSTPd120.ncl	
conccnSTPd14.ncl	
conccnSTPd3.ncl	
conccnSTPd5.ncl	
concnh4.ncl	
concno3.ncl	
concoa.ncl	
concpm10.ncl	
concpm2p5.ncl	
concso4.ncl	
diamcnmode.ncl	
et.ncl	Evapotranspiration
evspsbl.ncl	Evaporation
hfds.ncl	Downward heat flux at sea surface
hfls.ncl	Surface upward latent heat flux (includes both evaporation and sublimation)
hus.ncl	Specific humidity
LW_CRE.ncl	Longwave cloud radiative forcing
lwp.ncl	Vertically integrated cloud water (liquid only)
mlotst.ncl	Ocean mixed layer thickness
mmraer.ncl	
mmrbcfree.ncl	
mmrbc.ncl	
mrso.ncl	
MyVar.ncl	
od550aer.ncl	
pr-mmday.ncl	Precipitation (total) in mm per day
pr.ncl	Precipitation (total)
psl.ncl	Surface pressure
rlutcs.ncl	TOA outgoing clear-sky longwave radiation
rlut.ncl	TOA outgoing all-sky longwave radiation
rsds.ncl	
rsutcs.ncl	TOA outgoing clear-sky shortwave radiation
rsut.ncl	TOA outgoing all-sky shortwave radiation
sic.ncl	Sea ice area fraction
sit.ncl	Sea ice thickness
so.ncl	Sea water salinity
sos.ncl	Sea surface salinity
stratospheric_column.ncl	
SW_CRE.ncl	Shortwave cloud radiative forcing
ta.ncl	Air temperature

tas.ncl	Near-surface air temperature
tauu.ncl	Surface eastward wind stress
tauv.ncl	Surface northward wind stress
tauw.ncl	Surface wind stress
to.ncl	Sea water temperature
tos.ncl	Sea surface temperature
total_column.ncl	
toz.ncl	
tropospheric_column.ncl	
tropoz.ncl	
ts.ncl	Skin temperature
ua-200-850.ncl	Wind u-component at 200 hPa and at 850 hPa (monsoon diagnostics)
ua-200.ncl	Wind u-component at 200 hPa
ua-850.ncl	Wind u-component at 850 hPa
ua.ncl	Wind u-component
uo.ncl	Sea water x velocity
va-200-850.ncl	Wind v-component at 200 hPa and at 850 hPa (monsoon diagnostics)
va-200.ncl	Wind v-component at 200 hPa
va-850.ncl	Wind v-component at 850 hPa
va.ncl	Wind v-component
vmrc2h4.ncl	
vmrc2h6.ncl	
vmrc3h6.ncl	
vmrc3h8.ncl	
vmrch3coch3.ncl	
vmrco_alt.ncl	
vmrco_azr.ncl	
vmrco_chr.ncl	
vmrco_eic.ncl	
vmrco_gmi.ncl	
vmrco_hpb.ncl	
vmrco_lef.ncl	
vmrco_mlo.ncl	
vmrco.ncl	
vmrco_nwr.ncl	
vmrh2o.ncl	
vmrnox.ncl	
vmro3.ncl	
vmro3_NE.ncl	
vmro3_NHext.ncl	
vmro3_NT.ncl	
vmro3_SE.ncl	
vmro3_SHext.ncl	
vmro3_ST.ncl	
vmro3_Trop.ncl	
vo.ncl	Sea water y velocity
wfpe-mmday	Water flux from precipitation and evaporation
zg.ncl	Geopotential height

Typically, all namelists are stored in the folder *nml*, the naming convention is *namelist_xxx.xml* with "xxx" being the name of the diagnostic and/or a description of the purpose of the namelist:

- **For papers**

  xxx = SurnameYearJournalabbreviation (e.g., stocker12jgr, stocker12sci1, stocker12sci2).

- **For copies of reports that are not publicly available**

  xxx = OrgYearTitleabbrev (e.g., unep10water, unep11gap, roysoc09geoengineering).

- **For grouped sets of diagnostics and performance metrics that do not follow a published paper or report**

xxx = an intuitive name describing the scientific topic (e.g., aerosol, MyDiag, SAMonsoon, SeaIce)

3.4 Standard header for the namelist

For the sake of documentation, standard headers are defined and applied to all namelists and scripts in the ESMValTool. This is a template of the standard header for the main namelist. The parts in red are the ones to be modified by the author.

```
<namelist_summary>
namelist_name.xml

Description
A one-sentence description of the namelist content and purpose.

Author(s)
Name Surname (Affiliation, Country - e-mail@address)

Contributor(s)
Name Surname (Affiliation, Country - e-mail@address)

Reference(s)
Reference to the paper(s) considered by this namelist (if available).
Author, N. et al., Journ. Abbrev., NN, P1-P2, doi: (YEAR)

This namelist is part of the ESMValTool.
ESMValTool project PI: Veronika Eyring (DLR, Germany - veronika.eyring@dlr.de)
</namelist_summary>
```

3.5 Example namelist

```
<namelist>
<namelist_summary>
# namelist_clouds.xml
#
# Description
# Diagnostics of clouds and hydrological cycle.
#
# Author(s)
# Axel Lauer (DLR, Germany - axel.lauer@dlr.de)
#
# Contributor(s)
#
# Reference(s)
#
# This namelist is part of the ESMValTool.
# ESMValTool project PI: Veronika Eyring (DLR, Germany - veronika.eyring@dlr.de)
</namelist_summary>

<GLOBAL>
  <write_plots type="boolean">          True          </write_plots>
  <write_netcdf type="boolean">         True          </write_netcdf>
  <force_processing type="boolean">     False         </force_processing>
  <wrk_dir type="path">                 work/         </wrk_dir>
  <plot_dir type="path">                work/plots/ </plot_dir>
  <climo_dir type="path">               work/climo/ </climo_dir>
  <max_data_filesize type="integer">    100           </max_data_filesize>
```

```
  <verbosity type="integer">              1            </verbosity>
  <exit_on_warning type="boolean">    False          </exit_on_warning>
  <output_file_type>                   ps            </output_file_type>
</GLOBAL>

<MODELS>
  <model> CMIP5_ETHZ CESM1-CAM5  Amon historical r1i1p1 2000 2004
/export/pa_data02/ESMVal/model/ETHZ_CMIP5/  </model>
  <model> CMIP5_ETHZ GFDL-ESM2G  Amon historical r1i1p1 2000 2004
/export/pa_data02/ESMVal/model/ETHZ_CMIP5/  </model>
  <model> CMIP5_ETHZ MIROC5      Amon historical r1i1p1 2000 2004
/export/pa_data02/ESMVal/model/ETHZ_CMIP5/  </model>
  <model> CMIP5_ETHZ MPI-ESM-MR  Amon historical r1i1p1 2000 2004
/export/pa_data02/ESMVal/model/ETHZ_CMIP5/  </model>
  <model> CMIP5_ETHZ NorESM1-M   Amon historical r1i1p1 2000 2004
/export/pa_data02/ESMVal/model/ETHZ_CMIP5/  </model>
</MODELS>

<!—
    This is an example of a comment in XML
 -->


<!-- Please do not change anything below this line,
   unless you want to modify the standard diagnostic settings. -->
<DIAGNOSTICS>
  <diag>
    <description> Cloud diagnostics</description>
    <variable_def_dir>        ./variable_defs/      </variable_def_dir>
    <variable>                  lwp                 </variable>
    <field_type>                T2Ms                </field_type>
    <diag_script_cfg_dir>    ./nml/cfg_clouds/    </diag_script_cfg_dir>
    <model> OBS UWisc sat v2 1988 2007 /export/pa_data02/ESMVal/obs/UWisc </model>
    <diag_script cfg="cfg_clouds.ncl">   clouds.ncl   </diag_script>
  </diag>
</DIAGNOSTICS>

</namelist>
```

4 Directory structure of the ESMValTool

An overview of the directory structure used in the ESMValTool is given in Table S9. This section summarizes the underlying principles of the structure.

- Common namelist settings (e.g., models, year ranges, diagnostics) are usually stored in one place.
- Less common settings may be hidden deeper in the directory structure (see also section 5).
- Diagnostic scripts that can be used by namelist entries are also stored in one place and it generally possible to combine them in a modular way (e.g., using the output of one routine as input for another).
- Reuse of code is strongly encouraged, i.e., one place for each functionality (modularity on the technical level).
- The goal is to centralize functionality in individual functions/procedures whenever there is the possibility of reusability. New developers are encouraged to consider building on or extending existing routines before introducing new ones.

- Routines are sorted into folders according to their functionality. Whenever possible, the hierarchy level of routines is reflected by their position in the directory structure.

5 Configuration files

5.1 nml/cfg_diag/cfg_diag*.typ

Diagnostic-specific settings can be passed via the configuration files. These are collected in the nml directory under subdirectories named like the corresponding diagnostic (e.g., *cfg_aerosol/*, *cfg_perfmetrics/*). The suffix ".typ" specifies the language the routine is written in.

There might be more than one configuration script per diagnostic set. All cfg_* files for a diagnostic set need to be in the same folder specified by the <diag_script_cfg_dir> entry of the namelist (*nml/namelist_*.xml*).

The configuration settings are specified as attributes of the variable "diag_script_info" (in NCL via "diag_script_info@attribute = ..." see example below). In order to activate these attributes, "diag_script_info" must be set to "True".

Example (NCL)

```
diag_script_info = True

diag_script_info@projection = "CylindricalEquidistant" ; map projection, e.g., Mollweide, Mercator
diag_script_info@styleset = "CMIP5" ; "CMIP5", "DEFAULT"
diag_script_info@colormap = "WhiteBlueGreenYellowRed" ; e.g., WhiteBlueGreenYellowRed, rainbow
diag_script_info@ncdf = "default" ; enable to output to netCDF; either use "default" or give a full file name
```

Example (Python)

```
class diag_script:
   def __init__(self):
      self.info = True
      self.color = 1
      self.factor = 2.0e-3
      self.name = "example"

diag_script_info = diag_script()
```

Example (R)

```
diag_script_info<-new()

diag_script_info[["begin_ref_year"]]<-1970
diag_script_info[["end_ref_year"]]<-2000
diag_script_info[["timescale"]]<-3
diag_script_info[["seasons"]]<-c("ann", "djf", "mam", "jja", "son")
```

6 Running the ESMValTool

The following section gives a brief description of the steps required by a user to run an existing diagnostic. As an example, the toy diagnostic *MyDiag* is chosen to illustrate the basic steps:

1. Check/edit the main namelist *nml/namelist_MyDiag.xml*:

a. If needed, set the pathnames in the <GLOBAL> section for the "work" directory (wrk_dir), the directory for the plots (plot_dir) and the directory for reformatted files (climo_dir). See section 3.1 for details and Table S1 for a complete list of variables in the <GLOBAL> section.

b. In the <MODELS> section, define the model(s) to be used, including the root path for the actual model data, e.g., *CMIP5_ETHZ MPI-ESM-LR Amon historical r1i1p1 2000 2004 /export/pa_data02/ESMVal/model/ETHZ_CMIP5/*. See section 3.2, Table S2 and Table S3 for details. The first year (here: 2000) and last year (here: 2004) or the model data processed for each model is specified in this section.

c. Optionally, change variable and the field type in the <DIAGNOSTICS> section. See section 3.3, Table S4 and Table S5 for details. An overview of the available "field types" is given in Table S6, Table S7 lists the available variables. Please note that the diagnostic section may include additional models and/or observational data.

2. Check/edit the configuration file *nml/cfg_MyDiag/cfg_MyDiag.ncl*. In case of the toy diagnostics *MyDiag*, you can for example change to map projection for the contour plot, by changing the value of the attribute diag_script_info@projection.

3. Run the ESMValTool (in the ESMValTool root directory): *python main.py nml/namelist_MyDiag.nml*

4. The output will be written to a subdirectory named like the diagnostics package (e.g., *MyDiag*) in the directories specified in the <GLOBAL> section of the namelist (step 1). The default directories are: *work/MyDiag* for the NetCDF output and *work/plots/MyDiag* for the plot(s) (see also Figure 3). Acknowledgements and references are written to the file *work/refs-acknows_MyDiag.txt*.

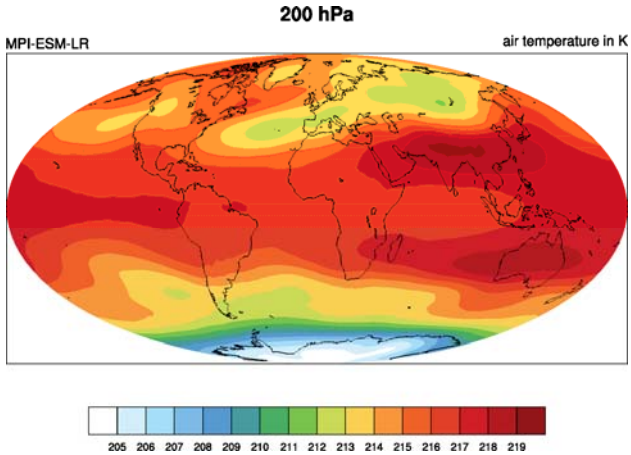


Figure 3 Example plot created by the toy diagnostic MyDiag showing the 5-year annual mean temperature at 200 hPa from the CMIP5 historical run (r1i1p1) of the MPI-ESM-LR model.

6.1 The acknowledgements log file

Each diagnostics in the tool automatically generates a log file containing a list of authors/contributors, details on the projects to be acknowledged and the reference papers to be cited. It also provides a list of the used model and observational data with the corresponding reference.

The log is created automatically when running the ESMValTool. The log file is named *refs-acknow_<diagnostics>*.txt and written to the directory defined in the <GLOBAL> section of the namelist (variable wrk_dir), e.g., *work/refs-acknows_MyDiag.txt* (see also section 6, step 4).

An example of an acknowledgements log file is provided below.

Example

```
---------------------------------------------------------------------------
++++++++++++ ESMValTool REFERENCES and ACKNOWLEDGEMENTS LOG ++++++++++++
---------------------------------------------------------------------------

Namelist file: nml/namelist_perfmetrics_CMIP5.xml
Creation date: Mi 25. Feb 09:25:13 CET 2015

Please acknowledge the use of the ESMValTool.
Please cite Righi et al., Geosci. Model Dev., 8, 733-768 doi:10.5194/gmd-8-733-2015, 2015.
For the specific diagnostics, see below.

---------------------------------------------------------------------------
=== perfmetrics_main.ncl ===

 Variable: ta

  Model: ERA-Interim
  Input path: /export/pa_data02/ESMVal/obs/ERA-Interim/
  Input file(s):
   (1) OBS_ERA-Interim_reanaly_1_T3M_ta.nc
  Fixes applied to input file(s): none

  Model: MPI-ESM-LR
  Input path: /export/pa_data02/ESMVal/model/ETHZ_CMIP5/historical/Amon/ta/MPI-ESM-LR/r1i1p1/
  Input file(s):
   (1) ta_Amon_MPI-ESM-LR_historical_r1i1p1_199001-199912.nc
   (2) ta_Amon_MPI-ESM-LR_historical_r1i1p1_200001-200512.nc
  Fixes applied to input file(s): none

AUTHOR(S):
 Frank, Franziska (DLR, Germany - franziska.frank 'at' dlr.de)

CONTRIBUTOR(S):
 Righi, Mattia (DLR, Germany - mattia.righi 'at' dlr.de)
 Eyring, Veronika (DLR, Germany - veronika.eyring 'at' dlr.de)
 Klinger, Carolin (DLR, Germany - carolin.klinger 'at' physik.uni-muenchen.de)
 Gottschaldt, Klaus-Dirk (DLR, Germany - klaus-dirk.gottschaldt 'at' dlr.de)

REFERENCE(S) FOR THIS DIAGNOSTIC:
 Please cite Righi et al., Geosci. Model Dev., submitted, 2014.
 Please cite Gleckler et al., J. Geophys. Res., 113, D06104, doi:10.1029/2007JD008972, 2008.
```

REFERENCE(S) FOR THE OBSERVATIONS:
NCEP - Kalnay et al., Bull. Amer. Meteor. Soc., 77, 437-470, 1996.
ERA-Interim
AIRS
CERES-EBAF
SRB

ACKNOWLEDGEMENTS FOR THE PROJECTS:
EU FP7 project EMBRACE
DLR project ESMVal

6.2 Model and observational data

When possible, observations from the obs4MIPs/ana4MIPs archives are used in the model evaluation. These data are freely available from the ESGF in the same format as the CMIP simulations and can be directly used in the ESMValTool using the **obs4mips** or **ana4mips** class in the namelist.

A collection of all observational data used by the diagnostics of the ESMValTool (trunk) is hosted at DLR and can be made available (restrictions by the data owner permitting) on request (see Table S8). The reformatted observational data can be read using the **OBS** class in the namelist.

All observations are tiered as follows:

- Tier 1: data sets from the obs4MIPs and ana4MIPs archives
- Tier 2: other freely available data sets
- Tier 3: restricted data sets (e.g., license agreement required)

Observational data sets not available in these archives need to be reformatted according to the CF/CMOR standard before they can be used (see section 8.2 for more details).

Part II: Developer's Guide

7 Writing a diagnostic script or a metrics set

The development of a new diagnostic (or set of diagnostics) requires the following steps before getting started:

- Creating a development branch in the applicable project subdirectory of the subversion repository (via svn, see section 12.1). Developers are encouraged to work actively through the subversion repository. Regular "commits" to the repository help to document changes introduced to the ESMValTool and allow for efficient sharing of code with other developers.
- Setting up a wiki documentation page for the new diagnostic/performance metrics following the template on the ESMValTool wiki (see section 12.3).
- Creating a standard namelist following the template on the ESMValTool wiki (see also section 3.4).
- If needed, opening an issue on the Mantis bug tracker (see section 12.2).

General coding rules and conventions:

- Regular updates of the development branch (svn merge, see also section 12) are strongly recommended in order to keep it synchronized with the trunk.
- Modularizing all diagnostic scripts as much as possible, using the general-purpose code in *lib/* and separating the diagnostic calculations from the plotting routines.
- Before creating new functions or procedures, it should be considered to use or extend the existing routines within *lib/*. Each header (see section 7.1) provides an overview of the already implemented functions and procedures.
- Functions and procedures specific to a given diagnostic shall go in the subdirectory diag_scripts/*aux/*<*diagnostic*> (see Table S9).
- Main namelist, diag_scripts, functions and procedures shall be documented within the respective file using the templates provided on the ESMValTool wiki (see also sections 3.4, 7.1 and 12.3).
- Each diag_script shall contain a call to the function *write_reference* (see section 0) in order to generate a respective acknowledgements log file (section 7.7).

The reintegration of the development branch into the trunk can only be done by the core development team (see section 9) who shall be contacted as soon as the branch is ready for integration into the trunk. Before contacting the core development team the following items should be checked:

- The new branch runs works with different configuration options.
- If the *lib/* routines have been modified, all the diagnostics using these routines have to be tested (see automated testing, section 7.9).
- The new code complies with the coding rules and standards (see section 7.5) and follows the ESMValTool directory structure (see Table S9).
- All authors, contributors and data are properly acknowledged and referenced in the acknowledgements log file (see section 7.7).
- If the new observational data are used, the scripts to "cmorize" these data shall also be made available and placed as *reformat_obs_*<*name*> into the folder *reformat_scripts/obs/*. Once the branch has been merged to the trunk, it shall be moved to the *branches/legacy/* subfolder.

7.1 Standard template

All (diagnostic) scripts and namelists in the ESMValTool are documented following the standards defined by templates (see section 3.4 for the namelist template). The following describes the standard header for diagnostics scripts. The parts in red are the ones to be modified by the author.

- The modification history is in reverse chronological order (i.e., most recent on top) and the last entry always contains the "written" statement (optionally with a statement such as "based on" if derived from existing code).
- The author of each entry in the modification history is indicated with the author id as given in the author list in the master reference file (*doc/MASTER_authors-refs-acknow.txt*, e.g., A_surn_na = surname, name).
- All lines should be limited to a maximum of 79 characters (see section 7.5). Exceptions can be made to improve the readability of the code.

```
;;#############################################################################
;; TITLE OF THE DIAGNOSTIC
;; Author: Name Surname (Affiliation, Country)
;; PROJECT-NAME project
;;#############################################################################
;; Description
;;   A short description of the diagnostic
;;   Additional description of the diagnostic
;;   Add more bullets if required
;;
;; Required diag_script_info attributes (diagnostics specific)
;;   att1: short description
;;         keep the indentation if more lines are needed
;;   att2: short description
;;
;; Optional diag_script_info attributes (diagnostic specific)
;;   att1: short description
;;   att2: short description
;;
;; Required variable_info attributes (variable specific)
;;   att1: short description
;;   att2: short description
;;
;; Optional variable_info attributes (variable specific)
;;   att1: short description
;;   att2: short description
;;
;; Caveats
;;   List possible caveats or limitations of this diagnostic
;;   Features to-be-implemented shall also be mentioned here
;;
;; Modification history
;;   YYYYMMDD-A_X4Y4: extended...
;;   YYYYMMDD-A_X3Y3: bug-fixed...
;;   YYYYMMDD-A_X2Y2: adapted to...
;;   YYYYMMDD-A-X1Y1: written.
;;
;; #############################################################################

load …
load …

begin
```

```
  ...
  ...
end
```

7.2 Library functions

The folder *diag_scripts/lib/* contains general purpose routines used by several diagnostic scripts, these library routines are grouped in subfolders by language, i.e.,

diag_scripts/lib/ncl

diag_scripts/lib/python

diag_scripts/lib/R

Library routines are grouped into individual files by topic, some examples for the NCL library routines are:

- *diag_scripts/lib/ncl/latlon.ncl:* routines to compute grid cell areas, weighted area averages, etc…
- *diag_scripts/lib/ncl/regridding.ncl:* routines interfacing the ESMF regridding functions in NCL
- *diag_scripts/lib/ncl/statistics.ncl:* statistical routines not (yet) implemented in NCL
- *diag_scripts/lib/ncl/style.ncl:* centralized control of NCL plot styles, e.g., defines line colors/dashes/thickness for each model name in CMIP5, based on the style files in *diag_scripts/lib/ncl/styles/*.

For further details on the library functions, see the project wiki or the documentation given in the header of the functions themselves (see section 7.1 for a template).

7.3 Plotting functions

The folder *plot_scripts/* contains general purpose routines used for plotting by the diagnostic scripts. The plotting functions should facilitate the separation of computing the diagnostic and displaying the result. To this end they should handle both the case when called directly from the diagnostic script (with data to visualize as an argument), and the case when the computed diagnostic is passed along as a netCDF file. These plotting routines are grouped in subfolders by language,

- *plot_scripts/ncl*
- *plot_scripts/python*
- *plot_scripts/R*

Each subfolder further groups the plotting routines into files by topic, e.g., for the NCL library routines:

- *plot_scripts/ncl/contour_maps.ncl:* interfaces NCL plotting routines for contour map plots, contour polar maps and adding markers to contour maps
- *plot_scripts/nc/scatterplot.ncl:* interfaces NCL plotting routines for of scatter plots

For further details on the plotting functions, see the project wiki or the inline documentation in the functions themselves.

7.4 Adding new variables

Adding new variables requires changes to *reformat_scripts/recognized_vars.dat* (section 7.4.1) and possibly also to *reformat_scripts/recognized_units.dat* (section 7.4.2). In addition, a new definition file *variable_defs/<varname>.ncl* is needed (section 7.4.3; see Table S7 for a list of currently available variable definition scripts). If the variable is a **non-derived** variable (explained in section 7.4.3) it also needs to be defined in a file named *reformat_scripts/cmor/CMOR_<variable>.dat* (see section 7.4.4)

7.4.1 reformat_scripts/recognized_vars.dat

New variables have to be added to *reformat_scripts/recognized_vars.dat*. Two lines are added per variable:

- std_name = varname
  standard CMOR variable name
- alt_name = alternative name 1, alternative name 2, …
  comma separated list of alternative variable names

Example (surface pressure)

- std_name = ps
- alt_name = aps,PS,psurf

The ESMValTool reformat scripts will look for variable “varname” in the input files. If not found, the alternative variable names “alternative name 1”, “alternative name 2”, etc. are tried before an error message is issued that the variable could not be found.

7.4.2 reformat_scripts/recognized_units.dat

The file *reformat_scripts/recognized_units.dat* contains a list of known units. If needed, the unit of the newly added variable can be added. There are two lines per unit:

- std_name = unit
  standard CMOR unit
- alt_name = alternative unit
  comma separated list of possible alternative units and corresponding conversion factor, defined as units[cmor] = units[alternative] * factor

Example (dobson units)

- std_unit = DU
- alt_unit = g m-2, 4.6707e-5, kg m-2, mol m-2, 2.2414e-3

7.4.3 variable_defs/varname.ncl

The file *variable_defs/<varname>.ncl* is a NCL script containing the declaration of the variable “varname” including its specific attributes. In case of derived variables, a function “calculate” calculating the derived variable must be defined in the script *<varname>.ncl* (see Table S7 for a list of currently available variable definition scripts).

Remarks

1. For derived variables, a statement specifying the (standard, non-derived) variables required to calculate the derived variable is needed. In the example given below, this statement in the beginning of the NCL script looks like

   *; Requires: rsut:T2*s,rsutcs:T2*s*

   In this example, the two standard variables “rsut” and “rsutcs” are needed to calculate the shortwave cloud forcing.

2. Variable attributes are specified as attributes of the variable “variable_info” (see examples below). In order to activate the variable attributes, “variable_info” must be set to “True”. Some examples for frequently used attributes are:

   - variable_info@derived = False (True)
   - variable_info@long_name = “…”
   - variable_info@units = “…”
   - variable_info@standard_name = “…”
   - variable_info@short_name =” …”

Example (precipitation, standard variable)

```
; Requires: none
variable_info = True
variable_info@derived = False
```

Example (shortwave cloud forcing, derived variable)

```
;  Requires: rsut:T2*s,rsutcs:T2*s

[…]

variable_info = True
variable_info@derived = True
variable_info@long_name = "CS Shortwave cloud radiation effect"
variable_info@units = "W m-2"

undef("calculate")
function calculate(index [1] : integer, variable [1] : string, field_type [1] : string)
;;              return_val [1] : logical
;; Arguments:
;;    index - index to current infile defined in the 'interface_data/ncl.interface'-file
;;    variable - Current variable as string
;;    field_type - string with field type classification
;; Return value:
;;    data_new – logical

local tmp, tmp1, tmp2, dum1, dum2, dum, i, verbosity
begin
   data_new = True
   tmp1 = read_data(index, "rsut", "T2Ms")
   tmp2 = read_data(index, "rsutcs", "T2Ms")
   dum1 = extract_data(index, tmp1, -1, 0, 0)
   dum2 = extract_data(index, tmp2, -1, 0, 0)

   dum = dum1
   dum = dum2 - dum1
   dum@long_name = variable_info@long_name
```

```
  dum@units = variable_info@units
  add_data_var(index, data_new, dum, variable)

  return(data_new)
end
```

7.4.4 reformat_scripts/cmor/CMOR_variable.dat

Each standard variable (non-derived) also needs a configuration file indicating the expected units of the variable. The expected units are read from the file *reformat_scripts/cmor/CMOR_variable.dat* which follows the definitions in the official CMOR tables for CMIP5. If this file is missing for a specific variable, it can be downloaded from http://pcmdi.github.io/cmor-site/tables.html. If a CMOR table for the new variable is not available, the user can create a new one based on the existing tables (e.g., following the example in *reformat_scripts/cmor/CMOR_mmrbcfree.dat* based on *reformat_scripts/cmor/CMOR_mmrbc.dat*).

Example, *reformat_scripts/cmor/CMOR_pr.dat*

```
SOURCE: CMIP5
!============
variable_entry:    pr
!============
modeling_realm:    atmos
!----------------------------------
! Variable attributes:
!----------------------------------
standard_name:     precipitation_flux
units:             kg m-2 s-1
cell_methods:      time: mean
cell_measures:     area: areacella
long_name:         Precipitation
comment:           at surface; includes both liquid and solid phases from all types of clouds (both large-scale and
convective)
!----------------------------------
! Additional variable information:
!----------------------------------
dimensions:        longitude latitude time
out_name:          pr
type:              real
valid_min:         0
valid_max:         0.001254
ok_min_mean_abs:   2.156e-05
ok_max_mean_abs:   3.215e-05
!----------------------------------
```

7.5 Coding rules and standards

The purpose of the code conventions used in ESMValTool is to ensure a high degree of consistency in the code layout. Consistently structured code increases readability and understanding of the code making it easier for developers and users work with a given piece of the code base. It is important to emphasize two points:

- Checking the code consistency should be done by software as this allows the check to be done automatically.
- Code checkers are available at *util/ncl-checker/pep8.py* (NCL) and *util/pep8-checker/pep8.py* (Python).

The code conventions are guidelines and should be treated as such. There are circumstances when it is advisable, for various reasons such as improved readability, to ignore some of the guidelines.

Code conventions used for Python

Python code should conform to the PEP-8 style guide [PEP8 2001]. Recommended tools to check Python code is the official PEP8-checker (*util/pep8-checker/pep8.py*) and PyFlakes. Further information on usage of these tools can be found on the ESMValTool wiki pages.

Code conventions for NCL

NCL code in ESMValTool should follow the PEP-8 style guides. An NCL adapted version of the Python PEP-8 checker is available in the ESMValTool repository (*util/ncl-checker/pep8.py*). See the ESMValTool wiki for further details. Please note that the NCL checker may report some false-positive (e.g., the reading symbol -> is not recognized as such).

Code conventions for R

The code conventions for R should conform to the formatting produced by the R parser tree. See the ESMValTool wiki for further details.

7.6 Documentation of Software

In order to ensure that all code can be maintained, all diagnostic packages must be well documented. It is the responsibility of the software developers to embed their documentation into the code and to provide a summary of their diagnostics on the ESMValTool wiki (see also section 7.8). Documentation systems exist to organize embedded documentation into well structured, linked documents.

- **R:** documentation should follow CRAN guidance.
- **Python:** the Sphinx package allows embedded documentation to be assembled into indexed web pages (see section 7.8)
- **NCL and namelists:** a Sphinx extension has been developed to extract code documentation for NCL and namelists (see section 7.8)

7.7 The acknowledgements log file

The acknowledgements log file automatically created by each diagnostic (see also section 6.1) is written by the function *write_references* (*interface_scripts/messaging.ncl*, see below), which uses the tags defined in the master reference/acknowledgements file (*doc/MASTER_authors-refs-acknow.txt*) as input. This master file lists all authors and contributors (tags starting with A_), the diagnostic references (tags with D_), references for observational data (tags E_) and projects (tags P_).

The function write_references

The function *write_references* (defined in *interface_scripts/messaging.ncl*) should be called at the end of each diagnostic script in order to write the acknowledgements log file (section 7.7). The function has the arguments “author(s)”, “contributors”, “diagnostics”, “observations”, “projects” which are arrays of strings. All strings (“tags”) used must be defined in the master reference file *doc/MASTER_authors-refs-acknow.txt*. The tags are then replaced by the function *write_references* with their definition when writing the acknowledgements log file. All tags in the master reference file are sorted by category of which there are four in total:

- A_xxx = authors, contributors (xxx = DLR user name)
  e.g., A_eyri_ve = Veronika Eyring

- D_xxx = diagnostics
  e.g., D_gleckler08jgr = Gleckler et al. (2008)

- E_xxx = observational data
  e.g., E_era40 = ERA40

- P_xxx = project
  e.g., P_embrace = EU FP7 project EMBRACE

```
write_references(diag_script, \
        "A_fran_fr", \
        (/"A_righ_ma", "A_eyri_ve", "A_gott_kl"/), \
        (/"D_righi15gmd", "D_gleckler08jgr"/), \
        (/"E_kalnay96bams", "E_erainterim", "E_airs", "E_ceresebaf", "E_srb"/), \
        (/"P_embrace", "P_esmval"/))
```

7.8 Documentation of source code

The Sphinx documentation generator (http://sphinx-doc.org) is used to organize and format ESMValTool documentation, including text which has been extracted from source code. Sphinx can help to create documentation in a variety of formats, including HTML, LaTeX (and hence printable PDF), manual pages and plain text.

Sphinx may be obtained from http://sphinx-doc.org/install.html; an overview of its workings is available at http://sphinx-doc.org/tutorial.html. In ESMValTool, Sphinx has been used to set up the files in *doc/sphinx*. Running *make* <*target*> in that directory will cause the documentation to be built, and its output placed in the *build/*<*target*> subdirectory. Here, <*target*> is the format required – for example, *html*, *latexpdf*, *man* or *text* for the four example formats mentioned above. Running *make* by itself will generate a complete list of output formats.

Sphinx was originally developed for documenting Python code, and one of its features is that it is able – using the so-called autodoc extension – to extract documentation strings from Python source files and use them in the documentation it generates. This feature apparently does not exist for NCL source files (such as those which are used in ESMValTool), but it has been mimicked (or – more-or-less – reverse-engineered) here via the Python script *doc/sphinx/scripts/process_ncl_docs.py*, which walks through a subset of the ESMValTool NCL scripts, extracts function names, argument lists and descriptions (from the comments immediately following the function definition), and assembles them in a subdirectory of *doc/sphinx/source*. These output files are in the so-called reStructuredText format (see, e.g., http://docutils.sourceforge.net/rst.html), which is the markup language used by Sphinx; running make in *doc/sphinx* builds the ESMValTool documentation from them, as noted above.

7.9 Automated testing

Any changes to a programming code have the risk of introducing unwanted side effects on some other parts of a code and introduce bugs. Routine and automated testing is therefore essential to maximize the code quality and ensure integrity of all diagnostics implemented within ESMValTool.

7.9.1 Setup and general workflow

Automated testing within the ESMValTool is implemented on two complementary levels:

- **unittests** are used to verify that small code units (e.g. functions/subroutines) provide the expected results

- **integration testing** is used to verify that a diagnostic integrates well into the ESMValTool framework and that a diagnostic provides expected results. This is verified by comparison of the results against a set of reference data generated during the implementation of the diagnostic.

Installation of the test environment

All scripts required to run the test environment are provided together with the ESMValTool code. Two external python packages are required which can be installed using the python package manager (pip; https://pypi.python.org/pypi/pip) as follows in a linux environment:

```
# install nosetests (https://nose.readthedocs.org/en/latest/)
pip install nose
# install easytest
pip install easytest
```

General functionality of testing framework

Each diagnostic is expected to produce a set of well-defined results. These are files in a variety of formats and types (e.g. graphics, data files, ASCII files …). While testing results of a diagnostic, a special namelist file is executed by ESMValTool which runs a diagnostic on a limited set of test data only. A small test data set is chosen to minimize executing time for testing while ensuring on the other hand that the diagnostic produces the correct results. The following general tests are implemented at the moment for diagnostics with available test data:

- **Check for file availability**: a check is performed that all required output data have been successfully generated by the diagnostic. A missing file is always an indicator for a failure of the program.

- **File checksum**: While the previous test only checks if a file is available, the checksum verifies if the content of a file is similar. Currently the MD5 checksum is used to verify that contents of a file are the same. The MD5 checksum is a good proxy for the similarity of two files and is used regularly to ensure integrity between files when transferring files between different computers.

- **Graphics check**: For graphic files an additional test is therefore implemented which verifies that two graphical outputs are identical. This is in particular useful to verify that outputs of a diagnostic remain the same after code changes.

Testing the ESMValTool diagnostics

Unittests are implemented for each diagnostic independently. Details on running unittests using **nose** is as simple as going to the ESMValTool root directory and then execute the following shell command:

```
# run nosetests
nosetests
```

This will search recursively for test files and execute these tests. A statistic on success and failures is provided at the end of execution. More details on using nose can be found in the package's documentation (https://nose.readthedocs.org/en/latest/).

To run integration tests for each diagnostic, a small script needs to be written once. An example for a file named `esmvaltooltest.py` is provided in section 7.9.2. To run all tests for diagnostics implemented in this file the following command needs to be executed:

```
# run integration tests
python esmvaltooltest.py
```

A summary of success and failures is provided as output.

7.9.2 Example test implementation for a diagnostic

In the following an example is given how to implement a test environment for a new diagnostic with just a few lines of code.

File: esmvaltooltest.py

```
"""
sample script for ESMValTool testing
"""

from esmvaltool import ESMValToolTest

"""
Define a new class for testing a particular diagnostic
"""

class PerfMetricCMIP5Test(ESMValToolTest):
    def __init__(self):
        # 1) define here the name of the test namelist
        nml_name = 'namelist_perfmetrics_CMIP5_test.xml'

        # 2) specify here the full path of the namelist
        # (relative to ESMValTool root)
        nml = 'nml/test_suites/dlr/' + nml_name

        # 3) define here the location of the reference data directory
        #    note that it is expected that the directory has the same
        #    name as the namelist
        refdir = esmval_dir + os.sep + os.path.splitext(nml_name)[0] + '/output/plots/'

        # initialize the parent class
        super(PerfMetricCMIP5Test,self).__init__(nml=nml, refdirectory=refdir,
esmval_dir=esmval_dir)

# ----------------------------------------

# This is how you run a test
PT = PerfMetricCMIP5Test()  # create instance of test class
PT.run_nml()   # run the testing namelist
```

```
PT.run_tests(execute=False, graphics=None,       checksum_files='all',files='all')  #
perform tests
```

8 Scientific documentation of a diagnostic script or metrics set

An important part of the implementation of a new diagnostic script is the documentation of the diagnostic on the ESMValTool wiki as well as the documentation of the observational data sets used. The former should comply with the standard template for new diagnostics (see section 8.1 below) and the latter should include instructions how to download the observational data and, if necessary, scripts to convert it to the format required in ESMValTool, see section 8.2 below.

8.1 Standard template

When implementing a new diagnostic script or metrics set, it should be documented on the ESMValTool wiki by creating a new wiki entry and filling out the below standard template:

Title of diagnostic / performance metric set	
Developers	*first name surname 1 (DLR tag 1), first name surname 2 (DLR tag 2), etc.*
Contributors	*first name surname 1 (DLR tag 1), first name surname 2 (DLR tag 2), etc.*
Date of documentation	yyyy-mm-dd
Name of standard namelist (XyZ)	For Xyz the following naming conventions is used: For papers XyZ=SurnameYearJournalabbreviation (e.g., stocker12jgr, stocker12sci1, stocker12sci2). For copies of reports that are not publicly available: XyZ=OrgYearTitleabbrev (e.g., unep10water, unep11gap, roysoc09geoengineering). For grouped set of diagnostics and performance metrics that do not follow a published paper or report an intuitive name that describes the science theme (e.g., XyZ=aerosol, MyDiag, SAMonsoon, SeaIce)
User settings	list of all settings that have to be checked/changed by a user in order to run the diagnostic (e.g., pathnames, configuration files, color tables, supported model names, etc.)
Brief summary	1-3 sentence summary
Status	Planned/work in progress/finished and merged to trunk
CMOR variable name (realm, frequency, dimension)	e.g., tas (atmos, monthly mean, longitude latitude plevs time)
Development branch	e.g., https://svn.dlr.de/ESM-Diagnostic/source/branches/CCMI/
Link to relevant Mantis issue	e.g., https://mantis.dlr.de/mantis/view.php?id=11742

1. Overview
   Insert text here
2. Available Diagnostics
   Insert text here
3. Specific Routines
   Contains a description of specific routines being developed for the given diagnostic that helps to identify common code (which should then go in the *lib/*)
4. Observations and Scripts (also see Model and observational data below)
   Insert text here
5. Test Cases (see also Automated testing on the project wiki and section 7.9)
   Insert text here
6. References
   - REF1:
   - REF2:
   - etc.
7. Sample Plots
   Please insert sample plots for all plot types produced by the namelist

8.2 Model and observational data

8.2.1 Overview

When possible, observations from the obs4MIPs/ana4MIPs archives are used in the model evaluation (see section 6.2). These data are freely available from the ESGF in the same format as the CMIP simulations and can be directly used in the ESMValTool using the **obs4mips** or **ana4mips** class in the namelist.

Observational data sets not available in these archives need to be reformatted according to the CF/CMOR standard before they can be used. In this case a reference to the official URL is provided such that a user can get the latest version of the data set as well as a description and a script how to convert the data set to the format required by the ESMValTool. These conversion scripts are collected in *reformat_scripts/obs/reformat_obs_<NAME>.ncl*. The reformatting routines must be documented with a standard header providing all information required to retrieve and process the data, as well as their availability (Tier 1, Tier 2, or Tier 3).

All observations are tiered as follows:

- Tier 1: data sets from the obs4MIPs and ana4MIPs archives
- Tier 2: other freely available data sets
- Tier 3: restricted data sets (e.g., license agreement required)

For Tier 2 and 3 data, the developer shall also provide links and helper scripts through the reformatting routines, following the template for the standard header described in section for the reformatting routines. An example can be found here:

https://svn.dlr.de/ESM-Diagnostic/source/trunk/reformat_scripts/obs/reformat_obs_AURA-MLS-OMI.ncl.

8.2.2 Standard header for the reformatting routines for observational data

This is a template of the standard header for the reformat_obs routines. The parts in red are the ones to be modified by the author. The modification history is given in reverse chronological order (i.e., most recent on top) and the last entry always contains the written statement. The author of each entry in the modification history shall be indicated with the author tag, as given in the master reference file (*doc/MASTER_authors-refs-acknow.txt*), e.g., A_surn_na = surname, name. All lines should be limited to a maximum of 79 characters.

```
;;#############################################################################
;; REFORMAT SCRIPT FOR THE OBSERVATION NAME OBSERVATIONAL DATA
;;#############################################################################
;;
;; Tier
;;    Information on data availability, possible options are:
;;    Tier 1: obs4MIPs or ana4MIPs
;;    Tier 2: other freely-available data set
;;    Tier 3: restricted data set
;;
;; Source
;;    URL to the data source or the reference
;;
;; Last access
;;    YYYYMMDD
;;
;; Download and processing instructions
;;    Short explanation on how to download and process the data
;;
```

```
;; Caveats
;;   List possible caveats or limitations of this script
;;   Features to-be-implemented shall also be mentioned here
;;
;; Modification history
;;   YYYYMMDD-A_xxxx_yy: extended...
;;   YYYYMMDD-A_xxxx_yy: written.
;;
;; ############################################################################

load ...
load ...
```

8.2.3 Data with available reformatting routines

Table S8 Observational data with available reformatting routines for use with the ESMValTool.

Name	Tier	Description	Variables	Type	Time range	Script name
AERONET	2	Aerosol optical depth at 550 nm	od550aer	Ground	1992-2012	reformat_obs_AERONET.ncl
ARGO	2	Ocean mixed layer depth	mlotst	Buoy	Climatology	reformat_obs_Dong08.ncl
AURA-MLS-OMI	2	Tropospheric column ozone	tropoz	Satellite	2005-2013	
AURA-TES	2	Ozone mixing ratio	vmro3	Satellite	2005-2009	reformat_obs_AURA-TES.ncl
CASTNET	2	Aerosol surface level concentrations	concso4, concno3, concnh4	Ground	1987-2012	reformat_obs_CASTNET.ncl
CIRRUS	3	Aerosol vertical profiles	mmrbc, mmrbcfree	Campaign	late Nov. 2006	reformat_obs_CIRRUS.ncl
CloudSat-CPR						
CONCERT	3	Aerosol vertical profiles	mmrbc, conccnSTP14	Campaign	-	reformat_obs_CONCERT.ncl
CR-AVE	3	Aerosol vertical profiles	mmrbc	Campaign	-	reformat_obs_CR-AVE.ncl
DC3	3	Aerosol vertical profiles	mmrbc	Campaign	-	Reformat_obs_DC3.ncl
EANET	2	Aerosol surface level concentrations	concso4, concno3, concnh4	Ground	2001-2005	reformat_obs_EANET.csh
EMEP	2	Aerosol surface level concentrations	concso4, concno3, concnh4, concpm2p5, concpm10	Ground	1970-2012	reformat_obs_EMEP.csh
Emmons	2	Vertical profiles of	various	Campaign	variable	reformat_obs_Emmons.csh

		gases				
EOC-GOME		Total column ozone		Satellite	1996-2010	
ERA-Interim	3	Basic climate parameters	ta, ua, va, zg, hus, tas, ps, tos, tauu, tauv	Reanalysis	1979-2012	reformat_obs_ERA-Interim.ncl
ERA-Interim fluxes	3	Basic climate flux parameters	pr, evspsbl, hfls, hfss, rsns, rlns	Reanalysis	1979-2012	reformat_obs_ERA-Interim-surffluxes.ncl
ESRL	2	CO2 surface level concentrations	co2	Ground	1973-2012	reformat_obs_ESRL.ncl
GLOBALVIEW		CO surface level concentrations	vmrco	Ground	1991-2008	
GPCC	2	Precipitation	pr	Reanalysis	1901-2010	reformat_obs_GPCC.ncl
GTO-ECV		Total column ozone	toz	Satellite	1996-2010	
HadCRUT	2	Near-surface air temperature	tas	Ground	1850-2013	reformat_obs_HadCRUT.ncl
HadISST	2	Sea ice concentrations and sea surface temperatures	sic, ts	Reanalysis	1870-2014	reformat_obs_HadISST
HALOE	2	Water vapor mixing ratio	vmrh2o	Satellite	1991-2002	reformat_obs_HALOE.ncl
HIPPO	3	Aerosol vertical profiles	mmrbc	Campaign	--	reformat_obs_HIPPO.ncl
IMPROVE	2	Aerosol surface level concentrations	concso4, concno3, concnh4, concbc, concoa, concpm2p5, concpm10	Ground	1988-2011	reformat_obs_IMPROVE.ncl
INCA	3	Aerosol vertical profiles	conccnSTP5, conccnSTP14, conccnSTP14	Campaign	--	reformat_obs_INCA.ncl
LACE	2	Aerosol size distributions	sizecn	Campaign	--	reformat_obs_LACE.ncl
LandFlux-EVAL	3	Evapotranspiration	et, et-sd	Synthesis product (model + observations)	1989-2005	reformat_obs_landflux-eval.ncl
MODIS-CFMIP	2	Ice water path	clivi	Satellite	2003-2014	reformat_obs_MODIS-CFMIP.ncl
Melpitz	2	Aerosol size	sizecn	Campaign	--	reformat_obs_Melpitz.ncl

		distributions				
NIWA	3	Total column ozone	toz	Reanalysis	1980-2010	reformat_obs_NIWA.ncl
NCEP	2	Basic climate parameters	ta, ua, va, zg, hus, tas	Reanalysis	1948-2012	reformat_obs_NCEP.ncl
NSIDC		Sea ice concentrations	sic	Satellite	1978-2010	
Putaud	2	Aerosol size distributions	sizecn	Campaign	--	reformat_obs_Putaud.ncl
SALTRACE	3	Aerosol vertical profiles	mmrbc	Campaign	--	reformat_obs_SALTRACE.ncl
SRB	2	Radiative fluxes	rsut, rlut, rlutcs	Satellite	1983-2007	Data from CCMVal2
TC4	3	Aerosol vertical profiles	mmrbc	Campaign	--	reformat_obs_TC4.ncl
Texas	3	Aerosol vertical profiles	mmraer, mmrbc	Campaign	--	reformat_obs_Texas.ncl
Tilmes		Ozone mixing ratios	vmro3	In-situ	1995-2009	
TRMM	2	Precipitation	pr	Satellite	1998-2004	
UB	2 (3 ?)	Sea-ice	sic	Satellite		
UCN-Pacific	3	Aerosol vertical profiles		Campaign	--	reformat_obs_UCN-Pacific.ncl
UWisc	3	Liquid water path	clwvi	Satellite	1988-2007	reformat_obs_UWisc.ncl
WOA09	2	Climatological ocean fields	so, sos, to	In-situ	Climatology	reformat_obs_WOA09.ncl

9 The ESMValTool core development team

9.1 Main Contacts

Please do not hesitate to contact Veronika Eyring (veronika.eyring@dlr.de) or Axel Lauer (axel.lauer@dlr.de) for general or technical questions on the ESMValTool or in case you have problems with access to Mantis (see section 12.2), svn (section 12.1), wiki (section 12.3).

9.2 Core Development Team

- Veronika Eyring (DLR, Germany), veronika.eyring@dlr.de

  ESMValTool Core PI and Developer: contact for requests to use the ESMValTool and for collaboration with the development team (all projects), and access to Mantis, svn, Wiki

- Axel Lauer (DLR, Germany), axel.lauer@dlr.de

ESMValTool Core Developer: contact for technical questions (all projects), and access to Mantis, svn, Wiki

- Mattia Righi (DLR, Germany), mattia.righi@dlr.de

  ESMValTool Core Developer: contact for technical questions (all projects)

- Martin Evaldsson (SMHI, Sweden), martin.evaldsson@smhi.se

  ESMValTool Core Developer EMBRACE: contact for general technical questions and technical implementation of SMHI led diagnostics (only for EMBRACE project)

Contacts for specific diagnostic sets are the respective authors, as listed on the corresponding Wiki pages and in the source code.

9.3 Merge requests

9.3.1 Workflow core development team

The following workflow followed by the ESMValTool core development team takes place whenever a developer requests the merging of a diagnostics set into the trunk:

1. Check that the developer submits a **standard namelist** that calls a set of diagnostics / metrics
2. Check that the related documentation on the wiki is compliant with **documentation templates for diagnostics and metrics sets** (see section 7.1).
3. Check that the code follows **coding rules and standard** (see section 7.5).
4. Check that a **namelist** is provided for **automated testing** that runs on 2-3 models and a small set of observations/reference model/idealized data
   - Verify that such a **reduced and small set of observations/reference model/idealized data** is delivered for each diagnostic that is called by the standard namelist
   - Verify that an **example plot + netCDF for automated testing** created with this reduced data set is provided for each diagnostic that is called by the standard namelist as a reference
5. Check that also the **full set of observations** is provided that allows a sophisticated scientific application of the (full) standard namelist
6. Check that the **observations are documented** on the ESMValTool Wiki and that a **reformat routine** is available in case the original source is not in CMOR standard
7. Run the **automated testing** with all available diagnostics
8. Iterate with developer(s) on points 1-7 until the above items are fulfilled and the reference plots for all standard namelists included in the trunk can be reproduced

9.3.2 Responsibilities of ESMValTool developers

1. Accept the **ESMValTool license** agreement.
2. Provide documentation on the wiki that is compliant with **documentation templates for diagnostics and metrics sets**.
3. Provide well documented code that follows the **coding rules and standards**.
4. **For each merge request** to implement a diagnostic set into the trunk.

Scientific analysis

- Provide the **code for all diagnostics and metrics** that are called.
- Standard namelist running on (if possible) all CMIP5 models and corresponding plots that are produced (for Wiki).
- Provide the **full set of observations** that allows a sophisticated scientific application of the full standard namelist list (indicate source and if applicable license issues).
- Provide **documentation for the observations** on the wiki and a **reformat routine** if the original source does not follow the CMOR standard.

Automated testing (see section 7.9)

- Provide **the code for automated testing** for the diagnostic set that should be integrated into the trunk.
- Provide a **namelist for automated testing**.
- Provide a **reduced and small set of observations/reference model/idealized data** for each diagnostic that is called by the testing namelist.
- Provide **NetCDF + example plots for automated testing** based on the reduced data set and the standard namelist as a reference.

5. **Name a contact person** providing (scientific) support for your diagnostics.

10 References

Flato, G., Marotzke, J., Abiodun, B., Braconnot, P., Chou, S. C., Collins, W., Cox, P., Driouech, F., Emori, S., Eyring, V., Forest, C., Gleckler, P., Guilyardi, E., Jakob, C., Kattsov, V., Reason, C., and Rummukainen, M.: Evaluation of Climate Models. In: Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, Stocker, T. F., D. Qin, G.-K. Plattner, M. Tignor, S.K. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex and P.M. Midgley (Ed.), Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA, 2013.

NCL (2014) The NCAR Command Language (Version 6.2.1) [Software]. (2014). Boulder, Colorado: UCAR/NCAR/CISL/VETS. http://dx.doi.org/10.5065/D6WD3XH5

PEP8 (2001) https://www.python.org/dev/peps/pep-0008/

Taylor, K. E., Stouffer, R. J., and Meehl, G. A.: An Overview of Cmip5 and the Experiment Design, B Am Meteorol Soc, 93, 485-498, 2012.

XML http://www.w3.org/TR/xml/www.gmail.com

11 Annex A – More tables

Table S9 Directory structure of the ESMValTool sorted by file type.

Namelists	
nml/namelist_XyZ.xml	Namelists for specifying general parameters, input data and diagnostics to run.
Configuration files	
nml/cfg_XyZ/cfg_XyZ_*.typ	Configuration files for diagnostic scripts. The suffix ".typ" specifies the language the routine is written in. Note: there is usually than one configuration script per diagnostic set.
Scripts	
main.py	Driver script controlling the overall program flow
diag_scripts/ *MyDiag.ncl* *SeaIce_polcon.ncl* *SAMonsoon.ncl* etc.	Directory containing all diagnostics called by the namelists. Supporting routines are placed in "diag_scripts/lib" under the subdirectory corresponding to the programming language used (NCL, Python, R).
aux/ <diagnostic>/	Functions and procedures specific to a given diagnostic are stored in the subdirectory diag_scripts/*aux/<diagnostic>*.
lib/ ncl/ *ensemble.ncl* *regrid.ncl* *statistics.ncl* *style.ncl* … python/ *ensemble.py* *regrid.py* *statistics.py* *style.py* … … for other languages (e.g., R)	Functions that are called by the diag_scripts, for example statistics.typ collects all statistical functions in a single file. When adding a new function, it must be added to list in the header.
plot_scripts/ ncl/ plotting1.ncl plotting2.ncl … python/ plotting1.py plotting2.py … … for other languages (e.g., R)	Plotting routines; files should have an intuitive name for their purpose. Data to be plotted may be passed to them directly or via netCDF files.
interface_data/	Inter-process communication, e.g., between Python and NCL/R, is done by sourcing NCL/R specific files updated on the fly in this folder. These intermediate files are based on templates files for different languages.
interface_scripts/	Routines called from the workflow manager script "*main.py*", mainly used to handle the control flow of the tool, e.g., parsing namelists, updating temporary files in the folder *interface_data/*, etc).
reformat_scripts/	Routines for checking or reformatting raw input data.
cmor/ CMOR_<var>.dat	Contains the CMOR tables, defined as plain-text files CMOR_<var>.dat, where <var> is the variable's standard name. Additional tables can be added by the users, e.g., from http://www2-pcmdi.llnl.gov/cmor/tables/.
default/ reformat_default_main.ncl reformat_default_func.ncl	Contains the reformat routines, in two NCL scripts.
ECEARTH/ reformat_ECEARTH_main.ncl reformat_ECEARTH_func.ncl names_ECEARTH.dat make_lsm3d.sc	Contains the EC-Earth/NEMO-specific reformat routines.

	EMAC/ reformat_EMAC_main.ncl reformat_EMAC_func.ncl names_EMAC.dat	Contains the EMAC-specific reformat routines.
	fixes/ <project>_<model>_fixes.ncl	Contains the user-defined, project- and model-specific fixes, defined as NCL scripts <project>_<model>_fixes.ncl. A template is also provided for the user to add more fixes.
	obs/	Contains specific reformat routines for "cmorizing" observational data.
	constants.ncl	Contains general-purpose functions and procedure, called by the default, the ECEARTH- and the EMAC-specific routines.
	recognized_units.dat	Provides a list of possible alternative units to the CMOR standard and the corresponding conversion factor. Can be extended by the user.
	recognized_vars.dat	Provides a list of possible alternative variable names to the CMOR standard names. Can be extended by the user.
	variable_defs/	Declaration of variables, variable specific attributes and calculation of derived variables
Data folders		
		The data folders are specified in *nml/namelist_**, and thus may be different from the defaults given here. These folders contain the output generated by the ESMValTool and are created on the fly if needed. Note that these folders do not need to be in the same directory as the source code. They can be arbitrarily specified in the namelist as path relative to the root path. Using symbolic links is another option to separate the actual data from the code.
climo/		Quality checked and derived netCDF files, reformatted from the original data.
plots/		Destination directory for graphics files.
work/		Miscellaneous files produced during run-time, e.g., optional netCDF output and *references/acknowledgements.*

Table S10 Workflow of reformat routines.

Control flow of reformat_default

The *reformat_default_main.ncl* script sets the global variables as defined in reformat.py (input and output paths, variable name and field, model name and ensemble, etc.) and then performs a list of operations calling various functions and procedures defined in *reformat_default_func.ncl*. The workflow is as follows:

- find grid type: the data can be defined on a standard rectilinear grid or on an irregular grid. In the latter case, the script does not modify the grid properties and additionally attaches the area field (the area weights) for the irregular grid to the output file. The location of the area file is typically defined as an entry in the namelist, for example by using the project class **CMIP5_gridfile** where the final entry is the full path to the area file, see Table S2.
- read variable: the selected variable is read from the input file. If the variable is not found, the reading function checks for possible alternative variable names (as specified in recognized_vars.dat), before issuing an error message.
- apply project- and model-specific fixes: if a fixing procedure is found in the fixes/ directory for the selected project and model, it is called at this point in order to apply the user-defined corrections to the data.
- create time-series: the variable is read for the selected time range (start_year-end_year) and a time-series is created.
- rank/field consistency: the consistency of variable's rank with the given field (T3M, T2Mz, T2Ms, etc.). A simple calculation of the zonal mean is performed in case a rank 4 variable is provided with T2?z field.
- check fill values: a default missing values is assigned if the variable does not have one. The function then looks for data values that might represent undefined missing values. Currently it considers: -999., -9999., -99999., -999999., 1.e15, -1.e34. Finally, the ESMValTool default missing values (1.e20) is assigned as a standard _FillValue to the variable.
- reformat time coordinate: the time coordinate is reformatted according to the CMOR standard. If a calendar attribute is not assigned, the standard is used. The consistency of the time-series with the selected time range is checked.
- reformat vertical coordinate (applies only to certain fields and to rectilinear grids): the vertical coordinate is assigned "Pa" units, converting from the most common alternative units (mbar, bar, hPa) if required. The ordering is set from top to bottom (monotonically decreasing).
- reformat latitude coordinate (applies only to certain fields and to rectilinear grids): the ordering is set from South to North (monotonically increasing).
- reformat longitude coordinate (applies only to certain fields and to rectilinear grids): the ordering is set from 0 to 360 degrees.
- check units: consistency of the variable's units with the CMOR standard is checked. The CMOR table for the selected variable must be available in the CMOR/ directory (an error message is issued otherwise). Units renaming and

conversion can also be performed, if the corresponding information is given in recognized_units.dat.

- set variable attributes: the CMOR standard attributes are assigned to the selected variable. The corresponding CMOR table must be available in the CMOR/ directory (an error message is issued otherwise).
- write output file: the variable reformatted according to the CMOR standard is written in the selected output file.
- add info for irregular grids (applies only to irregular grids): the area file of the irregular grid is added, this file may later be used for averaging.

Control flow of reformat_ECEARTH

This reformat procedure can be used to convert raw EC-Earth/NEMO files to a format that complies with the ESMValTool requirements. It performs the following additional operations compared with the default workflow:

- find_name: translate the EC-Earth/NEMO name to a CMOR name using the table names_ECEARTH.dat.
- sub_staggergrid: determine grid type (T, U, V) and add that information to the filename.
- mask_land: land points have the value 0 in the raw files, not a fill value (missing value). This routine sets land points (as in the landmask file) to fill values.
- rename_time: rename time variable from EC-Earth name to standard name and remove the attribute _FillValue.
- rename_lev: vertical coordinate name in raw files depends on grid, rename it to lev. Requires the external input table names_ECEARTH.dat.
- add_ijcoordvars: add i and j variables and assign them as coordinate variables.
- convert_units: unit conversions that cannot be handled by check_units.
- add_ECEARTHgrid_info: add ECEARTH grid info (lat, lon, areacello and grid sizes) to the output.

Control flow of reformat_EMAC

The workflow is similar to the default case, but some additional operations specific to the EMAC model are performed in addition:

- find messy version: the MESSy version with which the EMAC output has been generated is read from the data.
- find EMAC name: the EMAC name of the selected variable is found from the table in names_EMAC.dat (an error message is issued if not defined). For complex variables (i.e., variables not directly available as EMAC output but derivable from other EMAC variables), a user-defined recipe can be provided in *reformat_scripts/EMAC/recipes/EMAC_recipes_xxx.ncl* to derive it.
- check field consistency: reads from names_EMAC.dat file the list of allowed fields for the selected variable (for example is not possible to select total column ozone toz as T3M field).
- check vertical integration type (only for T2?s types): reads from names_EMAC.dat the option for the vertical coordinate (C for column integration, S for surface value).
- start the time loop: the EMAC output is assumed to be monthly-aggregated (monthly averages are optional). The data are read starting from January of the start_year to December of the end_year.
- extract variable: the selected variable is searched in the EMAC output. If multiple files for a given month/year combination contain the selected variable, the following priority list is applied: time coordinate matching the field type (monthly mean or daily output), data from tracer_gp and tr_* streams/channels, first file in the list. For complex variables, the corresponding user-defined recipe is called (*reformat_scripts/EMAC/recipes/EMAC_recipes_xxx.ncl*). For T2?z types, the data are interpolated on constant pressure levels (defined in *reformat_scripts/constants.ncl*).
- create time series: within the time loop, a time series start_year-end_year is created.
- reformat coordinates, check units, set variable attributes and write output: these operations are applied exactly as in the default case.

The user can extend the *reformat_scripts/EMAC/recipes/EMAC_recipes_xxx.ncl* in order to calculate additional (derived) variables not directly available in EMAC.

12 Annex B – subversion, Mantis, wiki

12.1 Subversion repository

The ESMValTool development currently uses a subversion repository for managing and maintaining the source code.

The ESMValTool repository is located at: https://svn.dlr.de/ESM-Diagnostic/
(use https://svn.dlr.de/viewvc/ESM-Diagnostic/source/ for an improved web interface)

The following description gives an overview of the typical workflow and usage for implementing a new diagnostic into the ESMValTool. For general information on subversion see the online svn book at http://svnbook.red-bean.com/.

12.1.1 General do's and don'ts

Always use the "*svn rm <...>*", "*svn mv <...>*" and "*svn cp <...>*" instead of *rm*, *mv*, and *cp*. That way subversion will be able to keep track changes applied to the (local copy of the) development branch.

Note that it is possible to apply the above commands directly to the repository, e.g., "*svn copy <URL1> <URL2>*". This is useful when modifying folders higher up in the repository structure.

When merging, svn keeps track of the revisions merged via the svn:merginfo-metadata, i.e., it is (usually) not required to specify a revision range for a merge.

12.1.2 Typical workflow

NB: All of the following svn commands take the optional argument "*--username <USERNAME>*" to explicitly specify your (DLR) username.

1. Create a new branch by copying the latest version of the trunk into to your new branch on the repository, selecting the appropriate project (<your-project>) and choosing a new branch name (<your-branch>):

   svn copy -m "creating branch ..." https://svn.dlr.de/ESM-Diagnostic/source/trunk/ https://svn.dlr.de/ESM-Diagnostic/source/branches/<your-project>/<your-branch>

2. Checkout a local copy of your newly created branch (if you already have a <local-path>, you can just call svn update):

   svn checkout https://svn.dlr.de/ESM-Diagnostic/source/branches/<your-project>/<your-branch> <local-path>

3. Work in the local folder and commit your changes on a regular basis to the repository:

   svn commit -m "changed …"

4. Keep your development branch updated with the trunk, by bringing in trunk updates on a regular basis (recommended is at least once per week):

   svn merge https://svn.dlr.de/ESM-Diagnostic/source/trunk --dry-run

   The "--dry-run" option will give you a heads up to which files are about to be merged/updated and possible merge conflicts. NB: the --dry-run option does not display updates to the svn mergeinfo metadata. Updating the mergeinfo is always necessary and you should therefore, always, proceed with 5-c and 5-e even if the output from 5-a indicated no changes.

   svn merge https://svn.dlr.de/ESM-Diagnostic/source/trunk

5. Resolve any merge conflicts that were reported in step 4, then test the local changes (important) and commit your changes back to your branch with:

   svn commit -m "implemented the following changes: …"

NB: Always follow through with the *svn commit* even if no files were updated. The svn:merginfo has been updated and needs to be commited back to your branch.

6. Reintegrate your work to trunk by (core developers only, other users shall do a merge request to the core development team, see Responsibilities for ESMValTool Development Team):

   - switch your working folder to trunk: *svn switch https://svn.dlr.de/ESM-Diagnostic/source/trunk*
   - cd to the trunk (--dry-run means try operation but make no changes),
   - *svn merge --reintegrate https://svn.dlr.de/ESM-Diagnostic/source/<branch-URL>/feature-branch-name --dry-run*
   - do the actual reintegration: *svn merge --reintegrate https://svn.dlr.de/ESM-Diagnostic/source/<branch-URL>/feature-branch-name*

7. Test the changes locally (very important to keep the trunk stable), then commit. If a Mantis issue exists (see also section 12.2) it can be useful to refer to this in the commit comment.

   svn commit -m "merging feature-branch-name (Mantis #XXXXX) to the trunk"

8. At this point the reintegrated branch contains erroneous meta data and should not be used further. In order to prepare removal of the local copy, switch back to the feature branch:

   svn switch https://svn.dlr.de/ESM-Diagnostic/source/<branch-URL>/feature-branch-name

   Then either run

   svn rm <branch-URL>/feature-branch-name

   or, if it is worth to keep an easy access point to the branch in the repository, move it to the legacy folder:

   svn mv -m "moving feature-branch-name into legacy" <branch-URL>/feature-branch-name <branch-legacy-folder>

   Finally remove your local copy with

   svn update

   and - if needed - manually delete files that were not under version control.

12.1.3 Other common svn commands

- *svn status*: check status of changes to local files, e.g., files modified, moved, gone missing, etc.
- *svn status -uv*: verbose output (-v) + indicate updates in the repository with a '*' (-u)
- *svn info*: general info about the checked out files + details which repository URL the local files represent
- *svn update*: bring in current URL updates from the repository
- *svn diff*: check the difference between local edits and their original (= locally unmodified) state. Note: this diff is never done towards the repository files, see '*svn revert*' below.

- *svn ls <URL>*: list content of the repository <URL>
- *svn revert <file>*: undo local changes, i.e., the ones shown by '*svn status*' and '*svn diff*'. The file is reverted to its original state. Note: the original state is the locally unmodified state, i.e., the state the file had at the time of the previous update / commit. If another developer has committed an update of the same file to the repository then the original state will differ from the repository state.
- *svn rm*: svn "aware" 'rm'-command
- *svn mv*: svn "aware" 'mv'-command
- *svn cp*: svn "aware" 'cp'-command

12.2 Mantis bug tracker

The ESMValTool developers currently use the Mantis bug tracker (http://www.mantisbt.org/) to discuss any kind of problems and open issues encountered with the ESMValTool. The ESMValTool Mantis page can be found at: https://mantis.dlr.de/mantis/my_view_page.php

12.3 Wiki

The latest information on the ESMValTool and diagnostics under development can be found on the wiki available at https://teamsites-extranet.dlr.de/pa/ESMValTool/Wiki/Home.aspx. All users and developers are strongly encouraged to frequently check the ESMValTool wiki for new information, contact data or observational data.