ESMValTool v2.0
Technical Overview

Mattia Righi

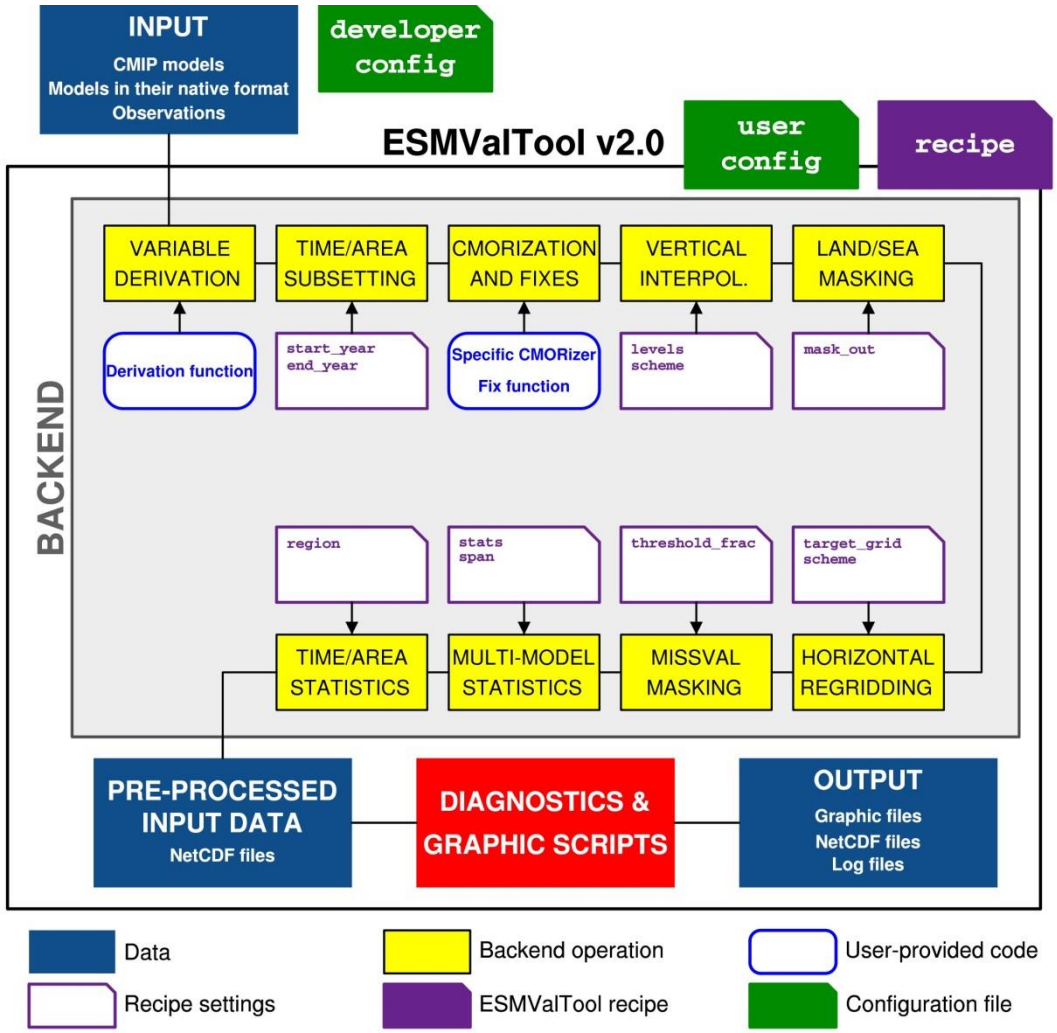
Version 16.11.2018

DLR

Outline

- Structure
- Installation
- Configuration
- Running ESMValTool
- Output directory structure
- Backend (aka preprocessor)
- Namelist, now called recipe
- Variable-specific and diagnostic-specific settings
- python-diagnostics interface
- Porting diagnostics v1.1.0 → v2.0
- Tips and tricks
- Development status and open issues (towards the Workshop…)

Structure


INPUT
CMIP models
Models in their native format
Observations
developer config
ESMValTool v2.0
user config
recipe
BACKEND
VARIABLE DERIVATION
TIME/AREA SUBSETTING
CMORIZATION AND FIXES
VERTICAL INTERPOL.
LAND/SEA MASKING
Derivation function
start_year
end_year
Specific CMORizer
Fix function
levels
scheme
mask_out
region
stats
span
threshold_frac
target_grid
scheme
TIME/AREA STATISTICS
MULTI-MODEL STATISTICS
MISSVAL MASKING
HORIZONTAL REGRIDDING
PRE-PROCESSED INPUT DATA
NetCDF files
DIAGNOSTICS & GRAPHIC SCRIPTS
OUTPUT
Graphic files
NetCDF files
Log files
Data
Backend operation
User-provided code
Recipe settings
ESMValTool recipe
Configuration file

Installation

- ESMValTool v2.0 is no longer a python script (`main.py`) but an actual python module and must therefore be installed
- Required software
  - anaconda3 or miniconda3 (available at https://conda.io/miniconda.html)
  - git (usually available via `module load git`; v2.0 or higher is recommended)
  - ESMValTool (https://github.com/ESMValGroup/ESMValTool.git, branch `version2_development`)
- Installation instructions (mind the shell `tcsh/bash`!)
  - https://github.com/ESMValGroup/ESMValTool/blob/version2_development/README.md
- The `setup.py` script creates a copy of the source code in the conda path and an `esmvaltool` executable
- For developers, it is advisable to install in development mode, which creates a link to (instead of a copy of) the source code in the installation path
- Conda allows to install and maintain multiple environments, e.g. two separate environments for ESMValTool v1.0 and ESMValTool v2.0

Configuration

- Two types of configuration files
  - `config-user.yml`: contains user-specific settings (input and output paths, logging, parallelization options, etc.)
  - `config-developer.yml`: contains detailed machine settings, like data directory structures, changes to this files are usually not needed by the users
- **IMPORTANT:** do not change the `config-user.yml` file on the repository. This is just an example, create and modify your own local copy!
- The path to the input data in this file is now project-dependent (e.g., CMIP5, OBS, obs4mips, etc.). A default can be also specified.
- Optionally a `drs` flag can be given to specify the directory structure. These flags are defined in `config-developer.yml`
  - The path to the data consists of two parts: `[rootpath]` (user's selection) + `[drs]` (machine-dependent, e.g. DKRZ, ETHZ, BADC…) → important for ESGF coupling!

Running ESMValTool v2.0

```
esmvaltool -c <config-user> <recipe>
```

- The `recipe` argument is mandatory
- The `esmvaltool` executable can be launched from any location on your machine, like any other software
- **IMPORTANT**: when parallelization is used (`max_parallel_tasks > 1` in `config-user.yml`) the memory usage can be large, especially for long recipes. In this case, it is strongly recommended to run the tool as a job script.
- Additional option `--max-datasets` to run a short version of the recipe with a few datasets without creating an extra recipe file

Output directory structure

- In v2.0, only a single output directory (`output_dir`) is specified by the user (in `config-user.yml`)
- A subdirectory `<recipe>_YYYYMMDD_HHMMSS` (UTC time!) is automatically created (→allows running multiple recipes simultaneously without conflicts!)
- The whole ESMValTool output (log files, pre- and post-processed NetCDF, graphics) is saved in this directory
- The output is automatically structured in different subdirectories:
  - `<output_dir>/<recipe>_YYYYMMDD_HHMMSS/preproc/` (backend output: preprocessed files)
  - `<output_dir>/<recipe>_YYYYMMDD_HHMMSS/work/` (diagnostic output: NetCDF files)
  - `<output_dir>/<recipe>_YYYYMMDD_HHMMSS/plots/` (graphic output: ps, pdf, png…)
  - `<output_dir>/<recipe>_YYYYMMDD_HHMMSS/run/` (log files, interface files)
- Each subdirectory is further organized by variable and diagnostic, to prevent possible data overwriting when using the parallelized mode
- **IMPORTANT**: do not set the output directory in your **`$HOME`**, as it can easily exceed your quota!

DLR

Revised backend for data preprocessing

- **CMORization**: as before (including dataset-specific fixes); model-specific CMORizers (e.g. EMAC, GFDL) not yet implemented
- **Variable derivation**: as before, but now in python
  - examples for `toz`, `lwcre` and `swcre` available
- **Level interpolation**
  - `levels`: single value (in Pa!), list of values, dataset name (use dataset coordinate), or `reference_dataset`
  - `scheme`: interpolation scheme (`linear` or `nearest`)
- **Land/Sea/Ice masking**: allows to mask out specific parts of a map based on the fx masking variables
  - `mask_out`: part of the grid to be set to missing (`land`, `sea` or `ice`)
- **Regridding**
  - `target_grid`: dataset name (use dataset grid), string `"LATxLON"` (e.g., `"1.8x2.5"`), or `reference_dataset`
  - `scheme`: regridding scheme (`areaweighted`, `linear`, `nearest` or `unstructurednearest`)

Revised backend for data preprocessing

- **Missing values masking:** implements the missing-value-masking method from the perfmetrics diagnostics
  - threshold_fraction: [0-1], filtering level along the time coordinate for miss val filtering
- **Multi-model statistics**
  - `span`: `full` (maximum time range of all models) or `overlap` (overlapping time range only)
  - `statistics`: `mean` and/or `median` (use list to specify both, i.e. `[mean, median]`)
  - `exclude`: exclude specific dataset(s) from the multi-model stats (dataset's name(s), or `reference_dataset`)
- **Time and space statistics**: region selection and average

- All operations, except CMORization, can be switched off (→useful when porting diagnostics, to reproduce the behaviour of v1.1.0)
- The order of operation is in principle fixed, although the code is designed to allow for a customized order (not yet implemented)

New recipe format YAML

- In v2.0, the recipes are written in YAML (http://yaml.org/)
- Sections and subsections are defined using 2-space indentations
- The v2.0 recipes contain 3 main sections:
  - `datasets`: same as the model section in v1, but `path` key no longer required
  - `preprocessor`: contains the backend settings
  - `diagnostics`: same as before, with some additions
- The `GLOBAL` section has been fully replaced by `config-user.yml`
- The new recipe does not contain any user- or machine-specific settings anymore → 100% machine-independent, fully portable!
- **IMPORTANT**: yaml is pretty strict regarding spaces/tabs, it is strongly recommended to use editor enhancements for this language (→ *Tips & Tricks*)

Diagnostic- and variable-specific settings

- Diagnostic- and variable-specific settings are now fully specified via recipe
- No `variable_defs/` files anymore!
- No `cfg_*.ncl` anymore!
- Single control point for all settings, further increases recipe portability!
- Various types of settings can be defined: logicals (`true`/`false`), strings, scalars, arrays (as python lists `[…,…]`)
- Settings can be propagated within a given recipe using mapping (reduces verbosity)
- Settings are automatically passed to the diagnostics via the interface file (→ *python-diagnostics interface*) as NCL attributes of `diag_script_info`
- **IMPORTANT**: there is no distinction anymore between diagnostic- and variable-specific settings, all settings are attributes of `diag_script_info`
- `variable_info` is now used to store the information in the variable subsection of the recipe (e.g., `reference_dataset`, `field`, etc.)

python-diagnostics interface

- The interface handling the communication between the python workflow and the (multi-language) diagnostics has been completely rewritten and simplified
- All required information are stored in a single (diagnostic- and variable-specific) interface file (→ allows for parallel execution of multiple recipes / diagnostics!)
  - `<output_dir>/run/<diagnostic>/<diag-script>/settings.ncl`
- `load "interface_scripts/interface.ncl"` to automatically load this information in the (NCL) diagnostic script (this also automatically loads all other interface scripts `logging.ncl`, `auxiliary.ncl`, `data_handling.ncl`)
- The interface information are stored as attributes in 5 (list of) logicals:
  - `diag_script_info`: diagnostic-specific settings
  - `variable_info`: variable information (`field`, `reference_dataset`, etc.)
  - `config_user_info`: configuration settings (`plot_dir`, `work_dir`, `output_file_type`, etc.)
  - `input_file_info`: input data settings (`fullpath`, `start_year`, `end_year`, `name`, `ensemble`, etc.)
  - `preproc_info`: preprocessor settings (to be added)
- An analogous file (`settings.yml`) is generated for python and R diagnostics

Porting diagnostics v1.1.0 → v2.0

`./doc/sphinx/source/developer_guide2/porting.rst`

- Convert recipe format from xml to yml
- Create a copy of the diag-script in v2.0 (note the new directory structure in `diag_scripts/` !)
- Check and apply renamings (as listed in the table): most of them are related to the revised interface
- Replace preprocessing functionalities with the backend: start with no preprocessing (only CMORization) and gradually implement level selection, regridding, masking, multi-dataset statistics, etc.
- Move diagnostic- and variable-specific settings from cfg-files and variable-defs to the recipe
- Test the recipe/diagnostic in the new version: compare the NetCDF output in preproc- and work-dir, and the graphics in plot-dir (use `cdo diff`, `ncdiff`, `diff`)
- Clean the code: adhere to codacy standards, use logging/error/warning functions!
- Update the documentation for your recipe/diagnostic.

DLR

Tips & Tricks

- Open a github issue to track your work and ask for support
  - example https://github.com/ESMValGroup/ESMValTool/issues/293
- Create a branch from `version2_development` and keep it up-to-date
  - `git checkout version2_development`
  - `git checkout -b version2_<your-branch>`
  - `git merge --no-ff --no-commit version2_development`
- Use editor enhancements (colors!), in particular for yml
  - see `./esmvaltool/utils/editor-enhancements`
- Git cheat-sheet: http://files.zeroturnaround.com/pdf/zt_git_cheat_sheet.pdf

Development status and open issues

- Current open issues: https://github.com/ESMValGroup/ESMValTool/issues
- We have defined a #RoadToRelease through 4 git-projects with increasing priority
  - Finalization of recipe_perfmetrics_CMIP5.yml: https://github.com/ESMValGroup/ESMValTool/projects/6
  - Release of v2.0-alpha: https://github.com/ESMValGroup/ESMValTool/projects/2
  - Release of v2.0-beta: https://github.com/ESMValGroup/ESMValTool/projects/3
  - Release of v2.0: https://github.com/ESMValGroup/ESMValTool/projects/4
- Most urgent issues are marked with the HIGH-PRIORITY label: https://github.com/ESMValGroup/ESMValTool/issues?q=is%3Aissue+is%3Aopen+label%3A%22HIGH+PRIORITY%22

Happy programming!