Write your own ESMValTool recipes and diagnostic scripts

Yanchun He (NERSC)

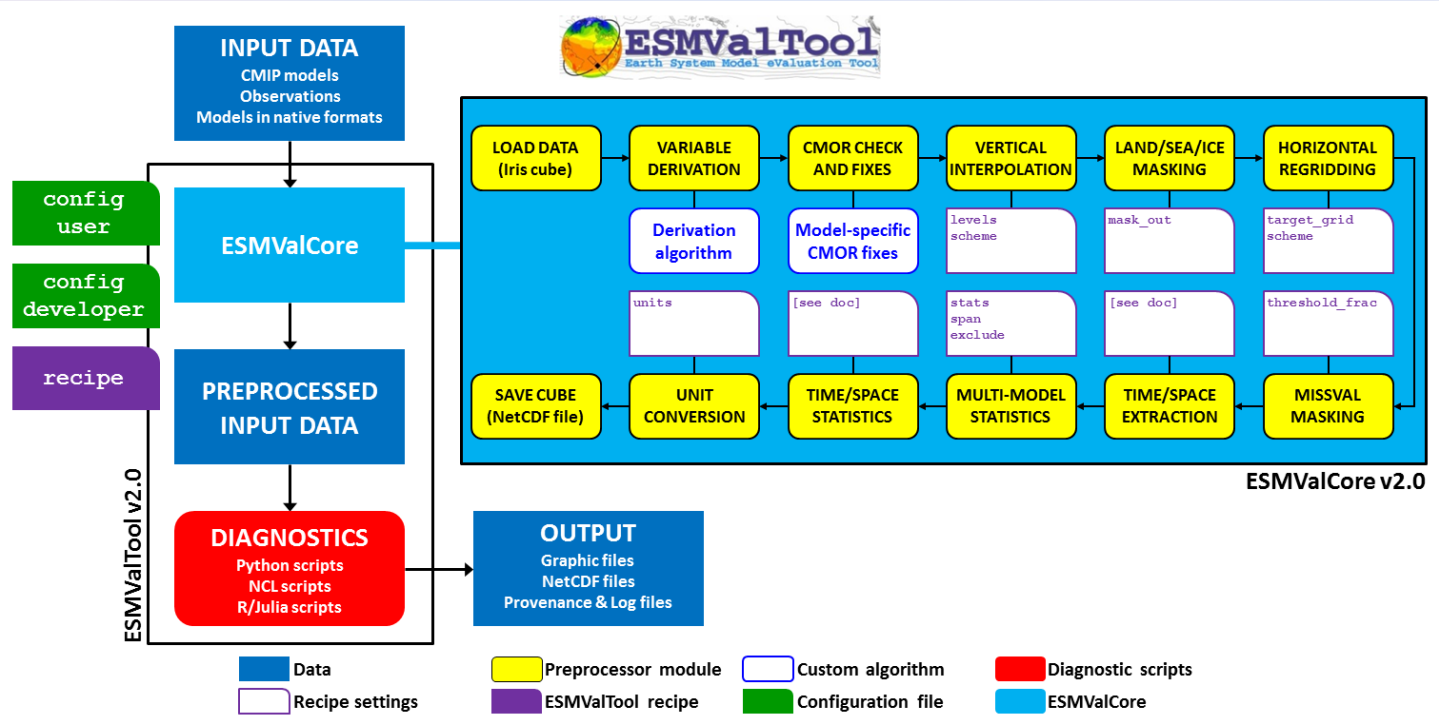
31th May, 2023

Commnunication between ESMValCore and ESMValTool

A complete ESMVALTool diagnostic procedure normally involves two steps:

- The ESMValCore read configuration, the recipes, and do the preprocessing, save the data and information as YAML files.
- The ESMValTool get information from the YAML files, and read the preprocessed data , and further pass information of the analaysed data so that the ESMValCore will save the data, plots, and their provenance.

- The commnunication between the ESMValCore and ESMValTool is through the saved YAML files (example here) and the diagnostic script interfaces. (more explainations on the interfaces).



Building a recipe from scratch

The easiest way to make a new recipe is to start from an existing one, and modify it until it does exactly what you need. However, in this episode we will start from scratch. This forces us to think about all the steps.

Specifically, this will cover:

- documentation
- datasets
- preprocessors
- diagnostics script

Detailed description of the recipe format is found at: "The recipe format":

The documentation

```yaml
documentation:
  title: Atlantic Meridional Overturning Circulation (AMOC) and the drake passage current
  description: |
    Recipe to produce time series figures of the derived variable, the
    Atlantic meridional overturning circulation (AMOC).
    This recipe also produces transect figures of the stream functions for
    the years 2001-2004.

  authors:
    - demo_le

  maintainer:
    - demo_le

  references:
    - demora2018gmd

  projects:
    - ukesm
```

See complet list of verified entries of `authors`, `maintainer`, `references` and `projects`.
https://github.com/ESMValGroup/ESMValTool/blob/main/esmvaltool/config-references.yml

Note: one has to add entries in the `config_references.yml` so that it can be used in the recipe. Use `unmaintained` as a general name before you change the `config_references.yml`.

Datasets

```
datasets:
  - {dataset: CanESM2, project: CMIP5, exp: historical, ensemble: r1i1p1, start_year: 2001, end_year: 2004}
  - {dataset: UKESM1-0-LL, project: CMIP6, exp: historical, ensemble: r1i1p1f2, start_year: 2001, end_year: 2004, grid: gn}
```

- dataset name (key dataset, value e.g. MPI-ESM-LR or UKESM1-0-LL).
- project (key project, value CMIP5 or CMIP6 for CMIP data, OBS for observational data, ana4mips for ana4mips data, obs4MIPs for obs4MIPs data, ICON for ICON data).
- experiment (key exp, value e.g. historical, amip, piControl, rcp85).
- mip (for CMIP data, key mip, value e.g. Amon, Omon, LImon). Also call `table_id`, see CMIP6 table_id.
- ensemble member (key ensemble, value e.g. r1i1p1, r1i1p1f1).
- sub-experiment id (key sub_experiment, value e.g. s2000, s(2000:2002), for DCPP data only).
- time range (e.g. key-value start_year: 1982, end_year: 1990).
- model grid (native grid grid: gn or regridded grid grid: gr, for CMIP6 data only).

Note: `start_year` and `end_year` are optional, as it will be included in the `diagnostic` section.

Preprocessors

```yaml
preprocessors:
 prep_map:
   regrid:
     target_grid: 1x1
     scheme: linear
   climate_statistics:
     operator: mean
   multi_model_statistics:
     span: overlap
     statistics: [mean]
```

More on the preprocessors

Diagnostics

A (simplified) example diagnostics section could look like

```
diagnostics:
  diagnostic_name:
    title: Air temperature tutorial diagnostic
    description: A longer description can be added here.
    themes:
      - phys
    realms:
      - atmos
    variables:
      variable_name:
        short_name: ta
        preprocessor: preprocessor_name
        mip: Amon
    scripts:
      script_name:
        script: examples/diagnostic.py
```

- The `title`, `description`, `themes` and `realms` entries are optional.
- The `diagnostic_name`, `variable_name`, `script_name` are customized but mandatory, and they will appear in the output files and directories.
- The `variable_name` can be the same as standard `short_name`, and then the `short_name` can be omitted (not tested)

Inside the diagnostic script

The diagnostic script will do some final fine-tunning analysis, visualise the results, save the data and the plots (by passing information back to ESMValCore). The structure of a diagnostic script is:

```
"""Python example diagnostic."""
import <some libraries>
from <libraries> import <method/function/class>

from esmvaltool.diag_scripts.shared import (
    run_diagnostic,
    ...
)
from esmvaltool.diag_scripts.shared.plot import quickplot

def some_function(xx):
    """ function for data analysis or visualisation, etc"""
    ...
    return

def main(cfg):
    ...
    some_function(xx)
    ...

    return

if __name__ == '__main__':

    with run_diagnostic() as config:
        main(config)
```

Inside the diagnostic script (cont.)

```
def main(cfg):
    ...
    some_function(xx)
    ...

    return

if __name__ == '__main__':

    with run_diagnostic() as config:
        main(config)
```

- The function `run_diagnostic` is called a context manager provided with ESMValTool and is the main entry point for most Python diagnostics. More on the shared diagnostic intefaces
- The commnunication between the ESMValCore and ESMValTool is through the saved YAML files (example here), passing to the diagnostic script by the context manager `run_diagnosc`. (more explainations on the interfaces).

Hands-on:

- https://nordicesmhub.github.io/esmvaltool-handson/03-write-a-simple-recipes-scripts
- https://esmvalgroup.github.io/ESMValTool_Tutorial/06-preprocessor/index.html
- https://esmvalgroup.github.io/ESMValTool_Tutorial/08-diagnostics/index.html

References

- https://docs.esmvaltool.org/projects/ESMValCore/en/latest/interfaces.html