ESMValTool – CHEAT SHEET

This cheat sheet is assuming access to an environment with *esmvaltool* installed. On Gadi the following commands will load the ACCESS-NRI *esmvaltool-workflow* to provide access.

```
module use /g/data/xp65/public/modules
module load esmvaltool
```

ESMValTool commands

Configuration
Before running a recipe, get the config-user file using the following command, which copies the installed version's config file to *~/.esmvaltool/config-user.yml* (~ is your HOME directory).

```
esmvaltool config get_config_user
```

When you run *esmvaltool* it will use this config file by default.
You can save a copy to another location to keep different settings with the `--path` option.

```
esmvaltool config get_config_user --path=<dest>
```

Open *config-user.yml* in a text editor (e.g. nano, vscode) to change settings. These are some main settings and options that you can check and change in your *config-user.yml* file:

```
output_dir: [<path> e.g.: /scratch/nf33/fc6164/esmvaltool_output]
remove_preproc_dir: [true/false]
download_dir: [<path>]
rootpath: […<paths> lists per project]
search_esgf: [never/when_missing/always]
```

The *config-developer*.yml can also be copied into the *~/.esmvaltool/* folder in a similar way to *config-user.yml*. This file has the directory structure settings used in the `drs: NCI` setting in *config-user.yml*

```
esmvaltool config get_config_developer
```

For example, CMIP6 would have these settings for directory structure in *config-developer.yml*:

```
CMIP6:
  cmor_strict: true
  input_dir:
    default: '/'
      …
    NCI: '{activity}/{institute}/{dataset}/{exp}/{ensemble}/{mip}/{short_name}/{grid}/{version}'
  input_file: '{short_name}_{mip}_{dataset}_{exp}_{ensemble}_{grid}*.nc'
  output_file: '{project}_{dataset}_{mip}_{exp}_{ensemble}_{short_name}_{grid}'
  cmor_type: 'CMIP6'
```

Recipe commands
View a list of available recipes with

```
esmvaltool recipes list
```

To copy a recipe into your current working directory use

```
esmvaltool recipes get <recipe file>
```

e.g. to copy *recipe_python.yml* into your working directory:

```
esmvaltool recipes get examples/recipe_python.yml
```

This can then be opened in a text editor, edited and run from that folder location or with an absolute path. Taking the above example, in your working directory with this edited recipe, you can use the following to run the edited recipe:

```
esmvaltool run recipe_python.yml
```

Data commands

View a list of observational datasets with CMORization functions available in *esmvaltool* with

```
esmvaltool data list
```

For info and downloading instructions

```
esmvaltool data info [DATASET]
```

For example:

```
esmvaltool data info WOA
```

The following command can be used to download observational datasets that have an auto-download available. Whether an auto-download is available can be checked with either of the above commands.

```
esmvaltool data download --config_file [CONFIG_FILE] [DATASET_LIST]
```

This following command can be used to CMORize the raw downloaded dataset.

```
esmvaltool data format --config_file [CONFIG_FILE] [DATASET_LIST]
```

The `--config_file` option is optional and can be used to specify a particular *config-user.yml* file. If you leave this option out, it will use the default *config-user.yml* file. For example:

```
esmvaltool data format NOAA-ERSSTv5
```

Observational data is downloaded in the location which is set in the *config-user.yml*. This is set in `RAWOBS` under `rootpath`.

```
rootpath:
  RAWOBS: ~/data/RAWOBS
```

This location is also used to find the raw data files to CMORize in the `data format` command. Formatted files will be found in the `output_dir:` setting in the *config-user.yml*.

Diagnostic recipe

An ESMValTool recipe aims to collate and find datasets of interest, run preprocessing on them so they are all ready for the diagnostics which can be written in a few different scripting languages. This outline shows a recipe has a standard tree (line numbers indicating each of these minimised).

```
  1   # ESMValTool
  2   ---
  3 > documentation: ···
 12
 13
 14 > datasets: ···
 17
 18 > preprocessors: ···
117
118 > diagnostics: ···
357
```

1. **documentation**

Minimum for documentation would be title, description, authors (as list). Under `documentation` can also be maintainer, references, projects, realms. etc.

```yaml
documentation:
  title: Model Monitoring
  description: |…
  authors:
    - vegas-regidor_javier
  maintainer:…
```

2. preprocessors

Multiple preprocessors can be defined with recipe author given name. e.g. (timeseries_regular, climatology…)

```yaml
preprocessors:
  timeseries_regular:…

  climatology:…

  climatology_pr:…

  climatology_500hPa:…
```

Each can have multiple preprocessor functions and their required parameters. e.g.

```yaml
  climatology_500hPa:
    extract_levels:
      levels: 50000
      scheme: linear
      coordinate: air_pressure
    climate_statistics:
      period: month
```

For available preprocessors refer to:
https://docs.esmvaltool.org/projects/ESMValCore/en/latest/recipe/preprocessor.html

3. diagnostics

Recipe author can give any name (here `plot_timeseries_annual_cycle`), and requires description, variables, and scripts. Can also have additional_datasets specific for the diagnostic.

```yaml
diagnostics:
  plot_timeseries_annual_cycle:
    description: "Plot time series and annualcycles"
    variables:…
    scripts:…
```

a) variables

There can be multiple variables under one diagnostic. Name of variable can be MIP 'short_name' which ESMValTool can interpret, or author named (here `nino3` and `nino34`) with `short_name` defined if using a different preprocessor which were defined in the `preprocessors` section e.g.

```yaml
    variables:
      nino3:
        plot_name: 'Niño 3 index'
        short_name: tos
        mip: Omon
        preprocessor: nino3
        grid: gn
      nino34:
        plot_name: 'Niño 3.4 index'
        short_name: tos
        mip: Omon
        preprocessor: nino34
        grid: gn
```

b) scripts

Can also have multiple and will be named. Can be *python, ncl, R, julia* and requires a `script:` key with value being the script file. Examples in recipe:

```
script: examples/diagnostic.R
script: seaice/seaice_yod.ncl
```

See *Scripts examples* section below for a quick view of the script files.
ESMValTool looks in installed *esmvaltool/diag_scripts/* for script file (above) or you can use an absolute path when writing your own. In this example `plt_script` is the name given by the recipe author and `colour` is a parameter used in the script.

```
96    scripts:
97      plt_script:
98        script: /home/189/fc6164/esmValTool/test_plotting.py
99        colour: blue
```

`scripts` can be set to null to not run a diagnostic script. This is useful for checking data can be found and used by ESMValTool or running preprocessors only.

```
32  diagnostics:
33    check_data:
34      description: check ESMValTool data
35    > variables: ···
49      scripts: null
```

Additional parameters which are used in the script can be defined below `script` key, e.g.

```
127     scripts:
128     plot: &plot_default
129       script: monitor/monitor.py
130       cartopy_data_dir: /g/data/xp65/public/apps/cartopy-data/shapefiles
131       plots:
132         timeseries: {}
133         annual_cycle: {}
```

4. datasets

Under `datasets`, which would be second after `documentation`, are lists used for all diagnostics and variables. These are given in key-value pairs or “facets” which define standardized data specifications.

```
13  datasets:
14    - {dataset: ACCESS-ESM1-5, activity: CMIP ,project: CMIP6,  grid: gn,
15       exp: historical, ensemble: r1i1p1f1, start_year: 1986, end_year: 2005}
```

Certain individual facets can also be defined in the variable or diagnostic. A required key is `project`,

```
- {project: [CMIP5/CMIP6/OBS/OBS6/ana4mips/obs4MIPs], ...}
```

Top level `datasets` key is not required if using `additional_datasets` under diagnostics or variables e.g.

```
87  diagnostics:
88
89    tas_change_all_models:
90  >   description: Air temperature change for RCP45 for all models as maps. ···
92  >   themes: ···
94  >   realms: ···
96  >   variables: ···
102 >   additional_datasets: ···
```

or

```
48      variables:
49        sic:
50          mip: day
51          preprocessor: extract_and_clim
52          reference_dataset: OSI-450-nh
53          additional_datasets:
54            - {dataset: OSI-450-nh, project: OBS, type: reanaly, version: v2,
55               mip: OImon, tier: 2, start_year: 1979, end_year: 2005}
```

YAML notes: some of these symbols are used in available recipes.

- `&` : use to save settings e.g.

```
plot: &plot_default
```

- `<<` , `*` : apply saved settings (called by name 'plot_default').

```
plot:
  <<: *plot_default
```

Saved settings can be altered by adding in a different key-value pair to overwrite saved settings.

Scripts examples

ESMValTool looks in installed *esmvaltool/diag_scripts/* for the script file given in a recipe which you can clone your own copy from https://github.com/ESMValGroup/ESMValTool or use an absolute path to your own script. More examples of scripts can be found in *esmvaltool/diag_scripts/* in your cloned copy.

Python

This is a brief example of a *python* diagnostic script with author functions minimised. Python libraries can be imported as well as *esmvaltool* helper functions, *run_diagnostic* is required here to get settings and datasets from recipe.

```
14   import logging
15   from esmvaltool.diag_scripts.shared import run_diagnostic
16   # This part sends debug statements to stdout
17   logger = logging.getLogger(os.path.basename(__file__))
18
19 > def plot_time_anomaly(ds,colour,longname,outname):···
47
48 > def compute_anom(data_dict): ##full and ref files···
59
60 > def rollingwindow(dataset):···
67
68 > def main(cfg):···
98
99   if __name__ == '__main__':
100      with run_diagnostic() as config:
101          main(config)
```

NCL

This snip of an *ncl* script (*esmvaltool/diag_scripts/seaice/seaice_yod.ncl*) shows the loading of some *esmvaltool* helper functions and the beginning of a routine to get you started.

```
33   load "$diag_scripts/../interface_scripts/interface.ncl"
34   load "$diag_scripts/../interface_scripts/logging.ncl"
35
36   load "$diag_scripts/seaice/seaice_aux.ncl"
37   load "$diag_scripts/shared/plot/aux_plotting.ncl"
38   load "$diag_scripts/shared/plot/scatterplot.ncl"
39
40   begin
41     enter_msg(DIAG_SCRIPT, "")
42     ; Get metadata items
43     att = True
44     att@mip = "OImon"
45     info = select_metadata_by_atts(input_file_info, att)  ; variable
46     var0 = info[0]@short_name
47     datasets = metadata_att_as_array(info, "dataset")
```

…

R

This is the start of an example *R* script (*esmvaltool/diag_scripts/examples/diagnostic.R*)

```r
library(tools)
library(yaml)
# get path to script and source subroutines (if needed)
diag_scripts_dir <- Sys.getenv("diag_scripts")
# source paste0(diag_scripts_dir,"/subroutine.r")
print(file.path("source ", diag_scripts_dir, "subroutine.r"))
# read settings and metadata files (assuming one variable only)
args <- commandArgs(trailingOnly = TRUE)
settings <- yaml::read_yaml(args[1])
```

…

Julia

This is a snip from the example *julia* script (*esmvaltool/diag_scripts/examples/diagnostic.jl*) with functions minimised to see an overall structure.

```julia
using PyPlot
# Avoid plotting to screen
pygui(false)

# Provides the plotmap() function
include(joinpath(dirname(@__DIR__), "shared/external.jl"))

function provenance_record(infile)…
end

function compute_diagnostic(metadata, varname, diag_base, parameter,…
end

function main(settings)…
end

settings = YAML.load_file(ARGS[1])
main(settings)
```

Outputs

Outputs for each recipe run are saved in the location defined in the *config-user.yml*. E.g.

```
output_dir: /scratch/nf33/fc6164/esmvaltool_outputs
```

The output is a folder named as the recipe file name and timestamp with a standard structure; plots, run, work, index.html in the root. The *index.html* can be used to view the output plots.

```
/scratch/nf33/fc6164/esmvaltool_outputs/depth_integration_20240228_031421
├── index.html
├── plots
│   └── diag_depthInt_1
├── run
│   ├── depth_integration_filled.yml
│   ├── depth_integration.yml
│   ├── diag_depthInt_1
│   ├── main_log_debug.txt
│   ├── main_log.txt
│   └── resource_usage.txt
└── work
    └── diag_depthInt_1
```

In the run folder, in *<diagnostic_name>/<script_name>* folder, log.txt is log output from the diagnostic script, e.g.

```
/scratch/nf33/fc6164/esmvaltool_outputs/depth_integration_20240228_031421/run/
├── depth_integration_filled.yml
├── depth_integration.yml
├── diag_depthInt_1
│   └── Global_Ocean_DepthIntegration_map
│       ├── diagnostic_provenance.yml
│       ├── log.txt
│       ├── resource_usage.txt
│       └── settings.yml
├── main_log_debug.txt
├── main_log.txt
└── resource_usage.txt
```

FAQ

Debugging

The *main_log.txt* and *log.txt* mentioned in the *Outputs* section above can be used for debugging.

Change datasets

Changing datasets in a recipe may require defining other facets, such as going from CMIP5 to CMIP6 `grid` will need to be defined, some variable name changes to be aware of, ensembles, and time ranges should also be considered, for example for different experiments. Observations will have other keys such as `tier, version, type.`

```
## replace datasets in original recipe:
  - {dataset: ACCESS-ESM1-5, activity: CMIP, project: CMIP6, grid: gn, exp: historical, ensemble: r1i1p1f1,
      start_year: 1986, end_year: 2005}
  - {dataset: ACCESS-ESM1-5, project: CMIP6, exp: piControl, ensemble: r1i1p1f1, grid: gn, start_year:  101, end_year:  250}
  - {dataset: ACCESS1-0, project: CMIP5, exp: historical, ensemble: r1i1p1, start_year: 1986, end_year: 2005}
  - {dataset: GPCP-V2.2, project: obs4MIPs, level: L3, start_year: 1986, end_year: 2005, tier: 1}
  - {dataset: ERA-Interim, project: OBS6, type: reanaly, version: 1, tier: 3}
```

CMIP6 and observational data on Gadi

NCI projects to join for CMIP6 data are `fs38, oi10.` There is a collection for Tier 1 and 2 observational datasets, the NCI project is `ct11`, these are CMORized so there is no need to download and CMORize these. The folder structures and are ready to be used with *esmvaltool* and the required `rootpath`s are set in the *config-user.yml* file that you get from the `esmvaltool config get_config_user` command.

References:
https://doi.org/10.5281/zenodo.3974592
https://docs.esmvaltool.org/en/latest/
https://github.com/ESMValGroup/ESMValTool