IS-ENES3 Deliverable D5.2

Technical standards for diagnostic tools

Reporting period: 01/01/2022 – 31/03/2023

Authors: Javier Vegas (BSC), Saskia Loosveldt Tomas (BSC), Kim Serradell (BSC), Stéphane Sénesi (IPSL-CNRS), Jerôme Servonnat (IPSL-CNRS), Bouwe Andela (NLeSC)
Reviewer(s): Klaus Zimmerman (SMHI)
Release date: 08/03/2022

ABSTRACT

This report presents a set of technical standards defined in order to facilitate the coupling of diagnostic scripts and packages with software frameworks designed to perform the evaluation and analysis of Earth System Models (ESM). These standards are proposed taking into account that diagnostics should interact with the evaluation frameworks in a modular and structured manner.

Revision table			
Version	**Date**	**Name**	**Comments**
Release for review	11/02/2022	Saskia Loosveldt Tomas, Kim Serradell	Initial version
First revision	01/03/2022	Klaus Zimmerman	Comments from first reviewer
Final version	08/03/2022	Sophie Morellon	Formatting
Dissemination Level			
PU	Public		X
CO	Confidential, only for the partners of the IS-ENES3 project		

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824084

Table of contents

Executive Summary

In the past years, efforts regarding the standardization to support the exploitation of data by the Earth system science community have focused on developing common European evaluation frameworks.

The adoption of these evaluation frameworks cannot be conducted without having an understanding of the users' requirements. Among these requirements, there is the need for evaluation tools to be interoperable, meaning that they should allow a modularized plug-in of diagnostic scripts.

This work introduces the definition of a standard in order to achieve this interoperability of diagnostic scripts across evaluation frameworks. The standard has been uploaded to Github, in order to make it public and allow for open and transparent discussion. The applicability of the current version of the standard has been tested with an example, which has also helped highlight how the standard could evolve in order to improve the already achieved interoperability.

1. Objectives

In Milestone 5.3, results were gathered from a survey conducted by IPSL and Assimila (WP3) and a workshop organized in Barcelona in May 2021 ("IS-ENES3 Virtual workshop on requirements for a fast and scalable evaluation workflow") in order to produce a list of user requirements for community evaluation tools.

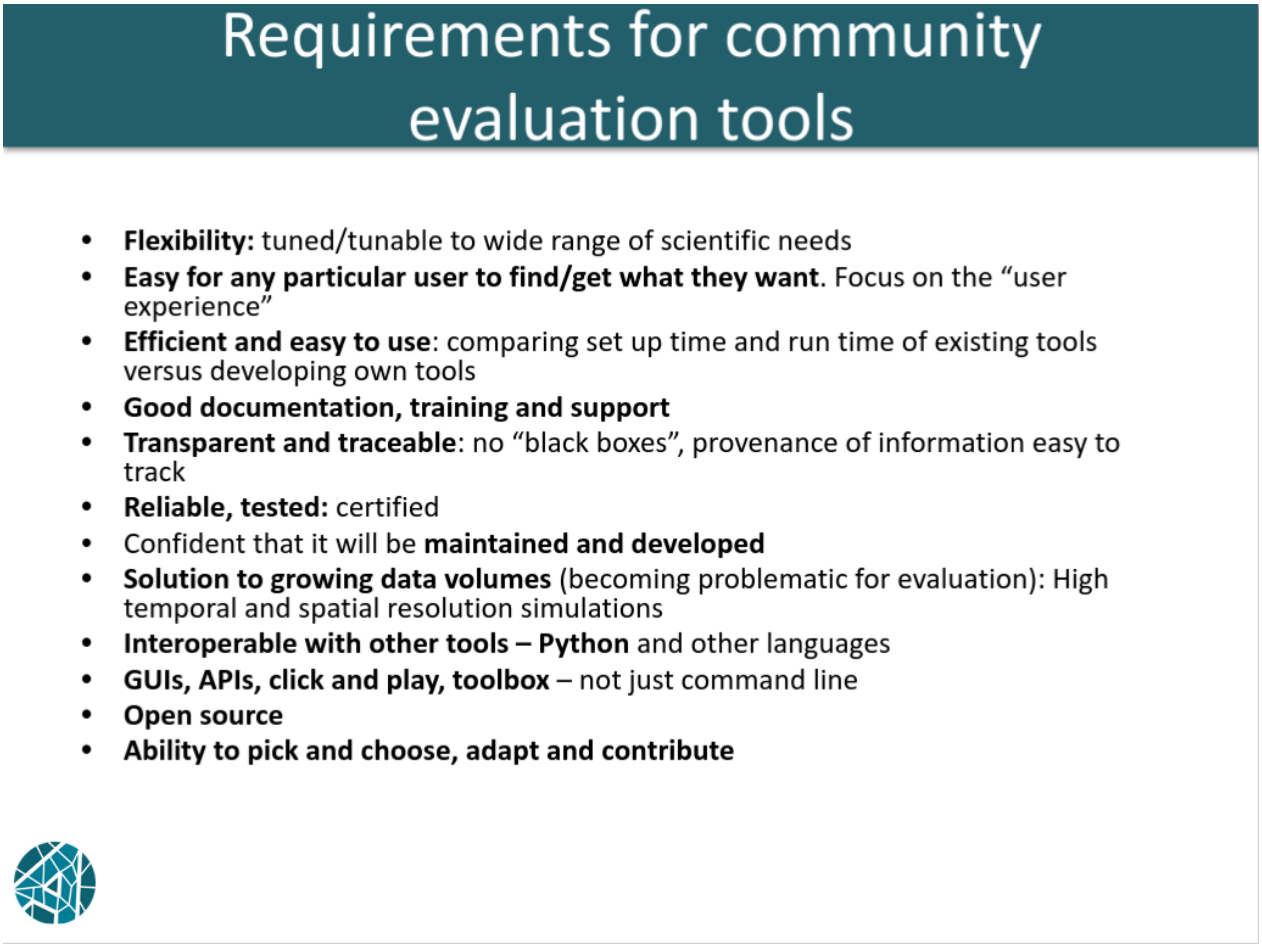

Figure 1. Summary of the requirements for community evaluation tools gathered from the IPSL and Assimila survey (2021)

Among these requirements, summarized in Figure 1, the work of this deliverable focuses on the requirement that evaluation tools must be "**Interoperable with other tools"**, both in **Python** but also in other languages. It needs to be noted that in the context of this deliverable, these **"other tools"** are understood as the set of codes that compute diagnostics and produce evaluation figures.

Since this interoperability must be achieved in a reproducible and shareable way, it requires the definition of a standard to describe the procedures that need to be followed in order to allow the coupling of diagnostics across evaluation tools.

In section 2, the standard will be presented and defined. After the definition of the procedures, an example of interoperability following this standard will be given in section 3.

2. Standard to achieve the interoperability of diagnostics across evaluation tools

The standard (Climate Diagnostic Interface Standard, CDIS) has been defined and is available in its own Github repository, which also hosts the set of coding guidelines that were delivered in D5.5 "Style guide on coding standards" Using a public website like GitHub to publish the standard is not a casual choice. We want this standard to be accepted by other tools, so it needs to be publicly available in a platform fostering discussions and tracking changes.

2.1. Definition of the standard

This document intends to define a standard with the purpose of facilitating the sharing of source codes used to analyze climate data and produce evaluation results and figures. Said codes will hereafter be referred to as 'diagnostic scripts', or in short 'diagnostics' or 'scripts'.

It is assumed that diagnostic scripts are able to be wrapped as standalone codes that run as independent processes, an approach that has been successfully tested in tools such as ESMValTool (Righi et al. 2020) and CLiMAF (version 2.0.2, 2022).

It also assumes that diagnostic scripts are able to read CF compliant NetCDF data files, such as the ones hosted on the Earth System Grid Federation (ESGF), and to produce output files with formats such as NetCDF, CVS, txt, or excel spreadsheets; as well as graphic files.

2.2. Glossary

This standard defines three tiers of enforceability for the guidelines to share diagnostics:

- **Must**: guidelines that all diagnostics have to meet to comply with the standard.

- **Recommend**: guidelines that all diagnostics are encouraged to meet, but are not mandatory. Nevertheless, if a list of compatible diagnostics is compiled, diagnostic scripts meeting recommended guidelines will be highlighted.
- **Suggest**: guidelines that are not required, but are considered to improve the quality of the diagnostics.

Diagnostic scripts *must* be contributed along with a YAML file containing configuration options (defined in section 2.3) and a set of files of the same format containing information regarding the input data (defined in section 2.4) to be used in the diagnostic.

In addition to these two files, it is *recommended* that another file of such format is provided as the script's formal description (defined in section 2.5).

The requirement of using YAML as the standard format for these files is motivated by the fact that it is common practice in software development to define configuration files in such a format. Furthermore, the syntax that it offers is human-readable and relatively straightforward.
Having easy-to-comprehend configuration and supporting files for the diagnostic script is essential in order to share them at a community level.
In terms of code quality, the availability of YAML linters to check the syntax is an added asset with respect to other file formats.

The standard also defines three different levels of enforceability for the contents of these YAML files:

- **Required**: parameters that must always be present, have a defined meaning and that can only take allowed values.
- **Reserved**: parameters that can be omitted, but if present, must have a defined meaning and allowed values.
- **Custom**: extra parameters can be defined as needed by the diagnostics or the tools. They must not collide with neither `required` nor `reserved` parameters.

2.3. The diagnostic configuration file

The diagnostic's configuration file *must* consist of a YAML file containing key-value mappings for all the parameters needed to run the diagnostic. Those parameters, defined below in terms of their enforceability level, will give information regarding: paths required for the execution of the diagnostic script and the storing of the generated output, the versions of the tool and diagnostic interfaces, and verbosity level of the logs generated by the script.

2.3.1. Required options

The following options *must* be present in the configuration file:

- **diagnostic_path** : `str`. Path to the diagnostic executable.
- **input_files**: `list`. List of absolute paths to the metadata files describing the data to be used by the diagnostic. The format of these files is specified in section 2.4.
- **tool**: `str`. Name of the tool used to prepare the data for the diagnostic.
- **version**: `str` Version of the tool used to prepare the data for the diagnostic.
- **run_dir**: `str`. Path to the directory to use as the current path for the diagnostic execution. *Must* be writable by the diagnostic.
- **data_dir**: `str`. Path to the directory to store the diagnostic's generated data. *Must* be writable by the diagnostic.
- **plot_dir**: `str`. Path to the directory to store the diagnostic's generated figures on. *Must* be writable by the diagnostic.
- **interface_version**: `str`. Version of the diagnostic interface.

It is *suggested* to provide different directory names for `**run_dir**`, `**data_dir**` and `**plot_dir**`, as it makes the interaction with the results easier. However, using the same directory for all purposes is accepted.

2.3.2. Reserved options

The following options are not required, however if present they must follow the definitions listed below:

- **log_level**: `str`, default value is `info`. Sets the granularity of the diagnostic logs and console output. If used, the logger must include `error` `warning`, `info` and `debug` levels .
- **auxiliary_data_dir** : `str`. Path to the auxiliary directory to store further input data files needed by the diagnostic, such as shapefiles (ESRI, 1998).

2.4. The data definition file

A data definition file *must* be provided for each variable required by the diagnostic. Each of these files *must* consist of a YAML file containing a two-level nested mapping.
The top-level mapping will have the path to the data file as keys and as values, the mapping containing the metadata related to the dataset and variable pairs, as described in the following sections.

2.4.1. Required options

- **alias**: `str`. Unique name for the dataset. Each (`alias`, `variable`) pair *must* be unique.
- **filename**: `str`. Absolute path to the data file.
- **variable**: `str`. Variable name to be used in the diagnostic. It is *not necessary* to follow CF conventions to name the variables.

2.4.2. Reserved options

2.4.2.1. Related to the dataset

- **project**: `str`. Name of the project to which a dataset belongs. It can be a real project, (e.g. `CMIP5`, `CMIP6`) or a tool-defined grouping of datasets (e.g. `Observations` or `Reanalysis`).
- **activity**: `str`. CMIP activity, or analog, of a dataset.
- **institute**: `str`. Name of the institute producing a dataset.
- **dataset**: `str`. Name of the dataset. For example: the CMIP model identifier, the name of an observational dataset, etc.
- **experiment**: `str`. Label to identify the experiment a dataset belongs to. (e.g `historical`, `piControl`).
- **ensemble**: `str`. Ensemble identifier of a dataset.

2.4.2.2. Related to the variable

- **table**: `str`. Variable's original MIP table.
- **original_frequency**: `str`. Variable's original frequency.
- **frequency**: `str`. Variable's frequency when entering the diagnostic.
- **modeling_realm**: `str`. Realms a variable belongs to.
- **grid**: `str`. If present, label of the variable's grid given by the CMIP6 grid label.
- **units**: `str`. Variable's units.
- **short_name**: `str`. Variable's name used in the file as given by the CMIP data request.
- **standard_name**: `str`. Standard name of the variable as given by CF conventions.
- **long_name**: `str`. Long name of the variable as given by the CMIP data request.

2.4.2.3. Custom

- **start**: `str`. Starting date of the data, given in ISO 8601 format for dates (YYYYMMDD).
- **end**: `str`, Ending date of the data, in ISO 8601 format for dates (YYYYMMDD).
- **start_year**: `int`. Start year of the data.
- **end_year**: `int`. End year of the data.
- **reference_dataset**: `str`. Alias of the dataset to be used as the reference for comparisons.

Therefore, if a diagnostic requires *N* variables [v1, v2, ... , vN] and *M* datasets [d1, d2, ..., dM] for each variable, the structure of the data definition files will be as follows:

data_definition_file_of_variable_v1.yml
- *path_to_dataset_d1.nc*
  - Required options for the pair (v1, d1)
  - Reserved options for the pair (v1, d1)
  - Other options for the pair (v1, d1)
- *path_to_dataset_d2.nc*
  - Required options for the pair (v1, d2)
  - Reserved options for the pair (v1, d2)
  - Other options for the pair (v1, d2)

…
- *path_to_dataset_dM.nc*
  - Required options for the pair (v1, dM)
  - Reserved options for the pair (v1, dM)
  - Other options for the pair (v1, dM)

data_definition_file_of_variable_v2.yml
- *path_to_dataset_d1.nc*
  - Required options for the pair (v2, d1)
  - Reserved options for the pair (v2, d1)
  - Other options for the pair (v2, d1)

…
- *path_to_dataset_dM.nc*
  - Required options for the pair (v2, dM)
  - Reserved options for the pair (v2, dM)
  - Other options for the pair (v2, dM)

...

data_definition_file_of_variable_vN.yml
- *path_to_dataset_d1.nc*
  - Required options for the pair (vN, d1)
  - Reserved options for the pair (vN, d1)
  - Other options for the pair (vN, d1)

…
- *path_to_dataset_dM.nc*
  - Required options for the pair (vN, dM)
  - Reserved options for the pair (vN, dM)
  - Other options for the pair (vN, dM)

Table 1. Example of the structure of a data definition file.

2.5. The script's formal description file

It is *recommended* that each script comes with a description file in YAML syntax.
The following labels should be provided in order to complete the documentation section:

- **title**: `str`. Label for the title of the diagnostic.
- **description**: `str`. A short description of the diagnostic.
- **references**: `list(str)`. List of references to be cited when using the diagnostic.
- **authors**: `list(str)`. List of authors that developed the diagnostic.
- **read_more**: `url`. An URL to the online documentation.

The labels listed below are *suggested* to provided in order to complete the formal description of the script:

- **script_name**: `str`. Name of the script, which has to be unique with respect to other scripts that can be run with the tool.
- **script_interface_version**: `str`. Label to specify the version of the script interface.
- **mandatory_keys**: `list(str)`. List of reserved and custom keys for which the tool *must* provide values. A schema should be provided in order to check the validity of the keys' values.
- **optional_keys**: `list(str)`. List of optional custom keys for the tool.

2.6. The command line interface

The diagnostic *must* be implemented as a command line tool that accepts the path to the YAML configuration file as a parameter.

It is *recommended* to include in the diagnostic executable the possibility to use a `--help` flag that prints the information defined in section 2.5

Additionally, it is possible to include additional flags to complement the execution of the diagnostics. For example, a flag could be provided when re-running a diagnostic in order to indicate whether the output directory should be emptied from previous runs or not.

3. Compatibility between the IS-ENES3 standard interface and CliMAF

The Climate Model Assessment Framework (CliMAF), is a natural candidate for being one of the tools using the CDIS We will comment on two levels of compatibility:

- Shallow compatibility: can CliMAF execute a script that follows the CDIS?
- Deep compatibility: does the CDIS affect some functionalities of CliMAF?

3.1. Shallow compatibility

CliMAF interface design is quite different from CDIS':

- it does not assume that the script is changed to adapt to the interface, and it adapts to the existing script command line structure;
- it communicates all script parameters and input data references through the command line, while CDIS uses the diagnostic configuration file;
- it treats separately data ensembles and single members, while CDIS handles single members as an ensembles of size one; part of the information provided in CDIS' script definition file is in CliMAF supported by the script declaration phase

Nevertheless, regarding feeding the script with input data and parameters, there is no basic incompatibility, as both interfaces have to provide the same basic set of information.

3.2. Deep compatibility

The most significant difference between CDIS and CLiMAF interface is about declaring script's outputs. CDIS provides no way to declare how many outputs the script will generate, nor to label them, while CliMAF needs such a declaration ahead of script execution. This is instrumental in CLiMAF's logic for caching all results, including in order to re-use outputs in further processing.

3.3. CliMAF changes needed for compatibility

The main changes that would be needed in CliMAF for interfacing with scripts matching the CDIS are:

- To create a function allowing both to declare an CDIS compatible script
- To create a corresponding Python function (called an CDIS-operator) usable in CliMAF scripting.

This operator would then allow to create a driver function able to :

- Build CDIS compliant master interface file and data definition files, based on datasets and keyword parameters used in calls to an CDIS-operator
- Create user-required working directories
- Call the script, taking care, if needed, of setting environment
- Return basic information on script success and results location

3.4. Compatibility proof-of-concept

In order to check the analysis above about CliMAF changes needed for CDIS compatibility, we took advantage of the fact that CDIS inherits from ESMValTool scripts interface, and implemented in version 2.0.2 of CliMAF the changes needed for interfacing with version 2.3 of ESMValTool, as described in the CLiMAF documentation.

```
# An example of declaring and calling an ESMValTool script from CliMAF

from climaf.api import *
from climaf.ESMValTool_diags import evt_script

# If your platform is not Ciclad, you must tell which is the wrapper for ESMValTool scripts
climaf.ESMValTool_diags.wrapper = \
    "/home/ssenesi/climaf_installs/climaf_running/scripts/"+\
    "ESMValTool_python_diags_wrapper_for_ciclad.sh"

# Create a CliMAF function for calling the ESMValTool diagnostic script
# (use the same syntax as the ESMVaTool recipe for designating the script)
evt_script("call_cvdp", "cvdp/cvdp_wrapper")

# Prepare input datasets for the diag.
base      = dict(project="CMIP6", experiment="historical",
                 realization='r1i1p1f2',  table="Amon", period="1850-1855", )
models    = [ "CNRM-CM6-1", "CNRM-ESM2-1"]

variables = [ "ts", "tas", "pr", "psl" ]

ensembles = []
for variable in variables:
    ensemble = cens(
        {
            model :  ds(model=model, variable=variable, **base)
            for model in models
        })
    ensembles.append(ensemble)

# Note : here, for other diagnostic scripts, you may have to reproduce
# the preprocessing steps that ESMValTool recipes implement upstream
# of the diagnostic script. For CVDP, there is actually no such
# preprocessing

# Call the diag. You may provide parameters that are known to ESMValTool
# or to the diagnostic script
wdir, prov = call_cvdp(*ensembles, output_dir="./out", write_netcdf=False)

# First returned value is the diag's working directory
print(wdir)

# Second one is a dictionnary of provenance information which
# describes all outputs (either graphics or NetCDF files) by various
# attributes, one of which being a 'caption'
one_output, its_attributes=prov.popitem()
print(one_output, its_attributes['caption'])

# But there is no further established framework in ESMValTool for a
# diagnostic to 'publish' a list of identifiers for its outputs
```

Figure 2. Example to declare and call an ESMValTool diagnostic script from CliMAF.

3.5. Changes to the standard needed for deep compatibility

The sole change in CDIS that is instrumental for CLiMAF to take full advantage of CDIS compatible scripts in CliMAF is related to the deep compatibility issue mentioned in section 3.2. It would be to include a mandatory script outputs declaration section; such a section would include, for each output :

- a label allowing CliMAF to name this output
- a pattern allowing CliMAF to find this output in one of the working directories

The need to include such a parameter in the definition of the standard will be discussed in Section 4, where the conclusions are presented.

4. Conclusions

A standard that defines a set of requirements and recommendations has been proposed in order to open the way for the sharing of diagnostic scripts across evaluation tools.

An example has been provided to showcase the applicability of this standard: A diagnostic script interfaced with the ESMValTool has also been interfaced with CliMAF following the guidelines defined in the standard.

While the parameters defined in the standard allow for a basic plug-in of the chosen ESMValTool script with CliMAF, it has been detected that the standard should be expanded to provide a definition for the output directories.

The definition of this parameter and its application to evaluation tools is the next immediate step needed in order to broaden the applicability of this standard.

The current version of the standard will be used to contact other groups developing evaluation tools to promote the adoption, increasing the interoperability of tools and therefore, giving to users a wider range of tools to perform strong and deep evaluation analysis.

References

Assimila 2021, *Community Survey of Model Evaluation Needs - Virtual workshop on requirements: Software requirements*,

Link: https://docs.google.com/presentation/d/1lgZRM9DsuE6qyBfs8oLXgMHZLf73Yvnl/edit#slide=id.p1

CNRM-GAME and IPSL, *CliMAF - a Climate Model Assessment Framework*, version 2.0.2, Link: https://climaf.readthedocs.io/en/master/index.html#

ESRI. 1998. *ESRI shapefile technical description* , Link: https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf

Righi, M., Andela, B., Eyring, V., Lauer, A., Predoi, V., Schlund, M., Vegas-Regidor, J., Bock, L., Brötz, B., de Mora, L., Diblen, F., Dreyer, L., Drost, N., Earnshaw, P., Hassler, B., Koldunov, N., Little, B., Loosveldt Tomas, S., and Zimmermann, K.: Earth System Model Evaluation Tool (ESMValTool) v2.0 - technical overview, Geosci. Model Dev., 13, 1179-1199, doi: 10.5194/gmd-13-1179-2020, 2020.