Reference documentation for ESMValTool

December 23, 2013

Contents

1 XML-configuration file

The XML-configuration file has the overall structure,

```
<namelist>
    <namelist_summary>
        A description of the current namelist
    </namelist_summary>
    <GLOBAL>
        <key1> value1 </key1>
        <key2> value2 </key2>
        <key3 attr="attr_value> value3 </key3>
        ...
    </GLOBAL>

    <MODELS>
        <model> entries to match data set 1 </model>
        <model> entries to match data set 2 </model>
        ...
    </MODELS>

    <DIAGNOSTICS>
        <diag>
            <description> Textual description of diagnostic</description>
            <variable_def_dir>  path  </variable_def_dir>
            <variable> variable1 </variable>
            <variable> variable2 </variable>
            ...
            <field_type> field_type1  </field_type>
            <field_type> field_type2  </field_type>

            <plot_script_cfg_dir> path  </plot_script_cfg_dir>

            <plot_script cfg="cfg_filename">  plot_script1.suffix
                                                    </plot_script>
            <plot_script cfg="cfg_filename">  plot_script2.suffix
                                                    </plot_script>
            ...
            <model> diag specific data set 1  </model>
            <model> diag specific data set 2  </model>
            ...
        </diag>

        <diag>
            ...
```

```
        </diag>
        ...
    </DIAGNOSTICS>
</session>
```

1.1 <namelist_summary>tag

To be written

1.2 <GLOBAL>tag

To be written

1.3 <MODELS>tag

To be written

1.4 <DIAGNOSTICS>tag

To be written

1.4.1 <diag>tag

To be written

2 Plot script auxiliary functions

To ensure some consistency across various plot scripts, which may be written in different languages altogether, this section recommends a number of "standard"-routines. The routines need to be implemented for each plot script language in use.

2.1 List of auxiliary routines/practices

2.1.1 Standard output handling

The python wrapper running a plot script collects its standard output/error, and, at the end of the plot script execution, writes this information back

to standard output. From the user side the verbosity of this output is controlled by the <verbosity>-tag in the namelist, an integer ranging from zero and upwards. A higher value indicates more verbose output. On the plot script side this is implemented as an 'info_output(...)'-function. Below is a minimalist NCL-version of this function, other script languages have similar functions implemented.

```
procedure info_output(output_string [1] : string,\
                      verbosity [1] : integer,\
                      required_verbosity [1] : integer)
begin
    if (verbosity .ge. required_verbosity) then
        print("info: " + output_string)
    end if
end
```

The purpose of this function is two-fold, to filter out messages with low priority (=high 'required_verbosity'), and to tag the message as an info message (to set it apart from warnings and errors).

2.1.2 Error handling

Because not all script languages used in ESMValTool propagates errors properly a proxy mechanism for error handling is in place. As the plot script exectution comes to an end the standard output/error is scanned for certain keywords indicating an error/warning was encountered during the execution. For NCL, these are,

- '`fatal:`' - indicates an error
- '`error:`' - indicates an error
- '`warning:`' - indicates a warning

If any of the error keywords are present, the script wrapper will will dump all output to standard output and raise an exception. Thus, to explicitly raise an error from an NCL plot script-routine it is necessary to use the construct,

```
print("fatal: NCL-error message")
```

Note that whether a `warning:` should halt execution is controlled via a switch in the xml-namelist (see above). Also note that most, but not all, native NCL-routines uses the prefix `fatal:` to indicate a critical error. If an error is suspected, but not catched by the `fatal:/error:` scan, it could be because a native NCL routine does not follow the `fatal:` convention, but prints different error message instead. In such cases, try increasing the verbosity such that all the plot script output is printed and check for any error manually.

The keywords for indicating errors/warnings are defined on a source code level, hence it is possible to use different keywords for other plot script languages.

2.1.3 Type of output

Many scripting languages supports different output formats, e.g., postscript, pdf, png, etc.. The user can indicate the preferred output type via the <`output_file_type`> in the namelist. Below shows the NCL-snippet handling this request in an NCL plot script,

```
file_type = getenv("ESMValTool_output_file_type")
if(ismissing(file_type)) then
    file_type = "PS"  ; Default output type
end if
```

Note that not all script languages supports all output types

2.1.4 Figure filenames

When the output from a plot script is a figure, this function provides a consitent naming of the output figure files. The main motivation for this routine is to simplify the search for output figures, or a specific set of output figures by following a naming convention. below is the declaration for the NCL version of this function.

```
output_filename = get_figure_outfile_name(plot_type, \
                                          variable, \
                                          field_number,\
                                          aux_title_info, \
                                          idx_mod)
```

¡¡¡¡¡¡¡ .working

2.2 Plot script template

Use the existing plot scripts and the `plot_scripts/MyDiag.ncl` as templates for creating new plot diagnostics.

3 ESMValTool control flow

To be written

4 ESMValTool data flow

To be written

5 Overview of available plot scripts

To be written. Currently this information is only available on the ESMValTool wiki page[3].

6 ESMValTool documentation and code conventions

This section proposes a recommended documentation and code convention practice within the ESMValTool.

6.1 Documentation of diagnostics

New diagnostics are implemented in branches of the subversion repository[2]. As part of the process of reintegrating them to trunk, they should be documented on the projected wiki[3]. This doucmentation should contain a technical description of the diagnostics, example of output (figures and/ord netCDF), relevant references if applicable, instructions on how to run the plot script. See existing plot script documentation for further details.

6.2 Inline documentation of scripts

6.2.1 Python

The recommended standard is PEP257[5].

6.2.2 NCL

The recommended standard is to follow the inline documentation used for the NCL functions in the online NCL-documentation[6]

6.2.3 R

The recommended standard is to follow the inline documentation used for the R functions at CRAN.

6.3 Code convention guidelines

The better part of the python routines in ESMValTool have been formatted according the Python PEP8 style[7]. The PEP8-style comes with a script for style guide checking, this script is available in a repository branch,

`https://svn.dlr.de/ESM-Diagnostic/source/branches/EMBRACE/pep8-compliance/util/pep8-checker/`

Additional python resources used for formatting code is the `redindent.py`-tool, available in the util folder of the same branch as is given above, and `pyflakes`[8].

6.3.1 NCL

The better part of the NCL functions have been formatted according to the Python standard PEP8[7]. A modified PEP8-checker, taking some of the NCL specific requirements into account, is available at,

`https://svn.dlr.de/ESM-Diagnostic/source/branches/EMBRACE/pep8-compliance/util/ncl-checker/`

The NCL-version is adaption of the Pyhton checker and works satisfactorily as long as one keeps in mind the false positives it finds due to language differences between Python and NCL. These false positives may be addressed in the future depending on priorities.

For consistent indentation across NCL files the emacs lisp script `ncl.el`[1], but with the default indent set to four instead of two (consistent with Python, see [7]). Using `ncl.el` requires comments to be written as '`;;`' to not be re-indented (which usually is not desirable). Note that `ncl.el` can be run via emacs batch mode and is therefore equally accessible for both emacs and non-ecmacs users.

6.3.2 R

The implemented R diagnostics have been reformatted using the "R parser tree"[1]. Note that this method can only be considered to semi-automatic since it does preserve comments (they need to be repatched) and does not produce very nice line breaks.

7 Testing

For testing purposes a Robot Framework[4] configuration file has been setup. The defined tests are on a namelist level, i.e., each namelist in the test case file is executed and the framework decides whether the test is passed or not depending on the existance and size of the expected output files from that namelist (figure/netCDF). Since running the tests requires namelists consistent with the platform at hand an individual setup is required for each platform. See the,

```
nml/test_suites/smhi/testcase_namelists.txt
```

file in the repository[2] for an example of a test case file. Running the test suite is done by,

```
pybot nml/test_suites/smhi/testcase_namelists.txt
```

References

[1] CRAN. Tidying-R-code. http://cran.r-project.org/doc/manuals/R-exts.html#Tidying-R-code.

[1] http://www.ncl.ucar.edu/Applications/Files/ncl.el

[2] ESMValTool. ESMValTool subversion repository, . https://svn.dlr.de/ESM-Diagnostic/source/trunk.

[3] ESMValTool. ESMValTool wiki page, . https://teamsites-extranet.dlr.de/pa/ESMValTool/Wiki/Home.aspx.

[4] Robot Framework. Robot Framework - a generic test automation framework. http://code.google.com/p/robotframework/.

[5] David Goodger and Guido van Rossum. Docstring conventions (pep257). Technical report, 2001. http://www.python.org/dev/peps/pep-0257/.

[6] NCAR. CISL's NCAR Command Language (NCL). http://www.ncl.ucar.edu/.

[7] Guido van Rossum and Barry Warsaw. Style guide for python code (pep8). Technical report, 2001. http://www.python.org/dev/peps/pep-0008/.

[8] Florent Xicluna. pyflakes - passive checker of Python programs. https://pypi.python.org/pypi/pyflakes.