ESM Tools r5.0 UserManual

Dirk Barbi, Nadine Wieters, Paul Gierz, Fatemeh Chegini, Miguel Andrés-Martínez, Deniz Ural

Jan 13, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

This is the user manual for the esm-tools. To contribute to this document, please contact the authors for feedback.

The esm-tools are a collection of scripts to download, compile, configure different simulation models for the Earth system, such as atmosphere, ocean, geo-biochemistry, hydrology, sea-ice and ice-sheet models, as well as coupled Earth System Models (ESMs). They include functionality to write unified runscripts to carry out model simulations for different model setups (standalone and ESMs) on different HPC systems.

CHAPTER

TWO

TEN STEPS TO A RUNNING MODEL

1. Make sure you have git installed with version newer than 2.13, that the python version is 3.6 or later (see *Before you continue*), and that pip is up-to-date (`pip install -U pip`). Also make sure that the location to which the python binaries will be installed (which is `~/.local/bin` by default) is in your `PATH`. For that purpose, add the following lines to one of your login or profile files, i.e. `~/.bash_profile`, `~/.bashrc`, `~/.profile`, etc.:

```
$ export PATH=$PATH:~/.local/bin
$ export LC_ALL=en_US.UTF-8
$ export LANG=en_US.UTF-8
```

2. Make sure you have a GitHub account and check our GitHub repository (`https://github.com/esm-tools`).

3. Download the git repository `esm_tools.git` from GitHub:

```
$ git clone https://github.com/esm-tools/esm_tools.git
```

4. In the new folder `esm_tools`, run the installer:

```
$ cd esm_tools
$ ./install.sh
```

   This should install the python packages of ESM-Tools. If you wonder where they end up, take a look at `~/.local/lib/python%versionnumber%/site-packages`. Also, a new file called `~/.esmtoolsrc` is added to your `HOME`, which contains some very few details about the installation.

5. Run `esm_master` once and answer the questions to setup the tool completely. You should see a long list of available targets if everything works. Note that you will need to manually edit the file `~/.esmtoolsrc`, if you mistakenly spelled any of the user names required for accessing the repositories, or you selected the default user name (`anonymous`).

6. Go to the toplevel folder into which you want to install your model codes, and run `esm_master install-`, followed by the name and the version of the model you want to install. As an example, if we want to install FESOM2:

```
$ mkdir ../model_codes
$ cd ../model_codes
$ esm_master install-fesom-2.0
```

   You will be asked for your password to the repository of the model you are trying to install. If you don't have access to that repo yet, `esm_master` will not be able to install the model; you will have to contact the model developers to be granted access (*Supported Models*). Feel free to contact us if you don't know who the model developers are.

Note: An error may occur in case you have performed a fresh install of` ESM-Tools` version 5 after having version 4 installed. In this known error, `esm_master` crashes with a `FileNotFoundError` with regard to `esm_master.yaml`. Try to fix this by updating your ~/.esmtoolsrc, removing lines that define paths for runscripts, namelists, and functions. Then try again (`RUNSCRIPT_PATH`, `NAMELIST_PATH` and `FUNCTION_PATH`).

7. Check if the installation process worked; if so, you should find the model executable in the subfolder `bin` of the model folder. E.g.:

```
$ ls fesom-2.0/bin
```

8. Go back to the `esm_tools` folder, and pick a sample runscript from the `runscripts` subfolder. These examples are very short and can be easily adapted. Pick one that is for the model you want to run, and maybe already adapted to the HPC system you are working on. Make sure to adapt the paths to your personal settings, e.g. `model_dir`, `base_dir` etc.:

```
$ cd ../esm_tools/runscripts/fesom2
$ (your_favourite_editor) fesom2-ollie-initial-monthly.yaml
```

   Notice that the examples exist with the endings `.run` and `.yaml`. It doesn't matter what you pick. The files ending in `.run` are looking more like conventional shell scripts that you might be better used to, the `.yaml`-files are written as yaml configuration files, which makes things much nicer and more elegant to write down. We strongly encourage you to give the `yaml`-version a try.

9. Run a check of the simulation to see if all needed files are found, and everything works as expected:

```
$ esm_runscripts fesom2-ollie-initial-monthly.yaml -e my_first_test -c
```

   The command line option `-c` specifies that this is a check run, which means that all the preparations, file system operations, ... are performed as for a normal simulation, but then the simulation will stop before actually submitting itself to the compute nodes and executing the experiment. You will see a ton of output on the screen that you should check for correctness before continuing, this includes:

   - information about missing files that could not be copied to the experiment folder
   - namelists that will be used during the run
   - the miniature `.sad` script that is submitted the compute nodes, which also shows the environment that will be used

   You can also check directly if the job folder looks like expected. You can find it at `$BASE_DIR/$EXP_ID/run_xxxxxxxxxxx`, where `BASE_DIR` was set in your runscript, `EXP_ID` (probably) on the command line, and `run_xxxxxxxxxxxx` stands for the first chunk of your chain job. You can check the work folder, which is located at `$BASE_DIR/$EXP_ID/run_xxxxxxxxxxx/work`, as well as the complete configuration used to generate the simulation, located at `$BASE_DIR/$EXP_ID/run_xxxxxxxxxxx/log`.

10. Run the experiment:

```
$ esm_runscripts fesom2-ollie-initial-monthly.yaml -e my_first_test
```

That should really be it. Good luck!

CHAPTER

THREE

INSTALLATION

3.1 Downloading

`esm_tools` is hosted on https://github.com/esm-tools. To get access to the software you need to be able to log into GitHub.

Then you can start by cloning the repository `esm_tools.git`:

```
$ git clone https://github.com/esm-tools/esm_tools.git
```

This gives you a collection of *yaml* configuration files containing all the information on models, coupled setups, machines etc. in the subfolder `config`, default namelists in the folder `namelists`, example runscripts for a large number of models on different HPC systems in subfolder `runscripts`, and this documention in `docs`. Also you will find the installer `install.sh` used to install the python packages.

3.2 Accessing components in DKRZ server

Some of the `esm_tools` components are hosted in the gitlab.dkrz.de servers. To be able to reach these components you will need:

1. A DKRZ account (https://www.dkrz.de/up/my-dkrz/getting-started/account/DKRZ-user-account).
2. Become a member of the group `esm_tools`. Either look for the group and request membership, or directly contact dirk.barbi@awi.de.
3. Request access from the corresponding author of the component. Feel free to contact us if you don't know who the model developers are or check the *Supported Models* section.

CHAPTER

FOUR

ESM TOOLS

For our complete documentation, please check https://esm-tools.readthedocs.io/en/latest/index.html.

4.1 Before you continue

You will need python 3 (possibly version 3.6 or newer), a version of git that is not ancient (everything newer than 2.10 should be good), and up-to-date pip (`pip install -U pip`) to install the *esm_tools*. That means that on the supported machines, you could for example use the following settings:

ollie.awi.de:

```
$ module load git
$ module load python3
```

mistral.awi.de:

```
$ module load git
$ module load anaconda3
```

glogin.hlrn.de / blogin.hlrn.de:

```
$ module load git
$ module load anaconda3
```

juwels.fz-juelich.de:

```
$ module load git
$ module load Python-3.6.8
```

Note that some machines might raise an error `conflict netcdf_c` when loading `anaconda3`. In that case you will need to swap `netcdf_c` with `anaconda3`:

```
$ module swap netcdf_c anaconda3
```

4.2 Installing

First, make sure you add the following lines to one of your login or profile files, i.e. `~/.bash_profile`, `~/.bashrc`, `~/.profile`, etc.:

```
$ export PATH=$PATH:~/.local/bin
$ export LC_ALL=en_US.UTF-8
$ export LANG=en_US.UTF-8
```

To use the new version of the esm-tools, now rewritten in Python, clone this repository:

```
$ git clone https://github.com/esm-tools/esm_tools.git
```

Then, run the `install.sh`:

```
$ ./install.sh
```

You should now have the command line tools `esm_master` and `esm_runscripts`, which replace the old version.

You may have to add the installation path to your `PATH` variable:

```
$ export PATH=~/.local/bin:$PATH
```

4.3 Configuration

If you have installed `esm_tools` you need to configure it before the first use to setup the hidden file `$HOME/.esmtoolsrc` correctly. This configuration will set required user information that are needed by both `esm_master` and `esm_runscripts` to work correctly. Such information are your user accounts on the different software repositories, your account on the machines you want to compute on, and some basic settings for the esm_runscripts.

To configure esm_master you should run the executable:

```
$ esm_master
```

Running it for the first time after installation, you will be asked to type in your user settings. This interactive configuration includes the following steps:

```
$ Please enter your username for gitlab.dkrz.de (default: anonymous)
$ Please enter your username for swrepo1.awi.de (default: anonymous)
```

Note that you will need to manually edit the file `~/.esmtoolsrc`, if you mistakenly spelled any of the user names required for accessing the repositories, or you selected the default user name (`anonymous`).

4.4 Uninstall ESM-tools

To uninstall your current installation of *ESM-Tools* you can use the following command:

```
$ esm_versions clean
```

You can also choose to manually uninstall. In order to do that, remove the installed Python packages and delete the `esm_*` executables. The following commands will do the trick if you installed with the `install.sh` script or installed using `pip` with `user` mode

```
$ rm ~/.local/bin/esm*
$ rm ~/.local/lib/python3.6/site-packages/esm*
```

Note that you may have a different Python version, so the second command might need to be adapted. You may also use `pip` to uninstall any of the packages:

```
$ pip uninstall [--user] esm-tools
```

The `--user` flag may be required when using `pip`.

CHAPTER

FIVE

TRANSITIONING FROM THE SHELL VERSION

5.1 ESM-Master

The Makefile based esm_master of the shell version has been replaced by a (python-based) executable called `esm_master` that should be in your PATH after installing the new tools. The command can be called from any place now, models will be installed in the current work folder. The old commands are replaced by new, but very similar calls:

```
OLD WAY:                          NEW WAY:
make                    -->       esm_master                    (to get the list of␣
↪available
                                                                 targets)
make get-fesom-1.4      -->       esm_master get-fesom-1.4      (download)
make conf-...           -->       esm_master conf-...           (configure)
make comp-...           -->       esm_master comp-...           (compile)
make clean-...          -->       esm_master clean-...          (clean)
```

Apart from that, the new esm_master offers certain new functionality:

```
esm_master fesom          (lists all available targets containing the string "fesom")
esm_master install-...    (shortcut for: get- , then conf- , then comp-)
esm_master recomp-...     (shortcut for: conf-, then clean-, then comp-)
esm_master log-...        (overview over last commits of the model, e.g. git log)
esm_master status-...     (changes in the model repository since last commit, e.g.␣
↪git status)
```

If the user wants to define own shortcut commands, that can be done by editing `esm_tools/configs/esm_master/esm_master.yaml`. New wrappers for the version control software can be e.g. added in `esm_tools/configs/vcs/git.yaml`. Adding commands in these configuration files is sufficient that they show up in the list of targets.

The details about models, setups, etc. are now to be found in `esm_tools/configs/esm_master/setups2models.yaml`. This file is a strucutred list instead of a barely readable, and rapidly growing, makefile. If you want to change details of your model, or add new components, this is where it should be put. Please refer to the chapter *ESM Master* for further details.

5.2 ESM-Environment

A visible tool, like esm-environment used to be, doesn't exist anymore. The information about the environment needed for compiling / running a model is contained:

- in the machine yaml file (e.g. `esm_tools/configs/machines/ollie.yaml`): This contains a default environment that we know works for a number of models / setups, but maybe not in an optimal way,
- in the model yaml file (e.g. `esm_tools/configs/fesom/fesom-2.0.yaml`): The model files are allowed to contain deviations from the default environment defined in the machine file, indicated by the keywords `environment_changes`, `compiletime_environment_changes` or `runtime_environment_changes`.

Please note that even though there still is a python package called `esm_environment`, this is just the collection of python routines used to assemble the environment. It does not contain anything to be configured by the user.

5.3 ESM-Runscripts

One main thing that has changed for the runtime tool is the way it is evoked:

```
OLD WAY:                                    NEW WAY:
./runscriptname -e experiment_id            esm_runscripts runscriptname -e experiment_id
```

Instead of calling your runscript directly, it is now interpreted and executed by the wrapper `esm_runscripts`, the second executable to be added to your PATH when installing the Tools. Internally, `esm_runscripts` reads in the script file line by line and converts it into a python dictionary. It is therefore also possible to write the "runscripts" in the form of a yaml file itself, which can be imported by python much easier. The user is invited to try the yaml-style runscripts, some example can be found in `esm_tools/runscripts`.

Some of the variables which had to be set in the script when using the shell version are now deprecated, these include:

- FUNCTION_PATH
- FPATH
- machine

Also the last two lines of the normel runscript for the shell version of the tools, `load_all_functions` and `general_do_it_all`, don't do anything anymore, and can be safely removed. They don't hurt though.

(. . . to be continued. . .)

5.4 Functions –> Configs + Python Packages

The shell functions, which used to be in `esm-runscripts/functions/all`, are gone. That was basically the whole point of re-coding the tools, to get rid of this mixture of model configuration, wild shell hacks, and in general lots of annoying problems. What used to be in the functions is now seperated into python code (which is actually doing things, but doesn't have any model-, setup- or machine specific information), and yaml configurations (which are basically structured lists of all the information we have, including mesh resolutions, scenario simulation forcings,. . .). Anything really that you could possibly know about running a simulation belongs into the yaml configs that you can now find in `esm_runscripts/configs`, while ESM-Tools functionality is coded in the python packages.

5.5 Namelists

No changes. Namelists can be found in `esm_tools/namelists`.

CHAPTER

SIX

YAML FILE SYNTAX

6.1 What Is YAML?

YAML is a structured data format oriented to human-readability. Because of this property, it is the chosen format for configuration and runscript files in *ESM-Tools* and the recommended format for runscripts (though bash runscripts are still supported). These *YAML* files are read by the *esm_parser* and then converted into a Python dictionary. The functionality of the *YAML* files is further expanded through the *esm_parser* and other *ESM-Tools* packages (i.e. calendar math through the *esm_calendar*). The idea behind the implementation of the *YAML* format in *ESM-Tools* is that the user only needs to create or edit easy-to-write *YAML* files to run a model or a coupled setup, speeding up the configuration process, avoiding bugs and complex syntax. The same should apply to developers that would like to implement their models in *ESM-Tools*: the implementation consists on the configuration of a few *YAML* files.

Warning: *Tabs* are not allowed as *yaml* indentation, and therefore, *ESM-Tools* will return an error every time a *yaml* file with *tabs* is invoked (e.g. *runscripts* and *config* files need to be *'tab-free'*).

6.1.1 YAML-Specific Syntax

The main *YAML* **elements** relevant to *ESM-Tools* are:

- **Scalars**: numbers, strings and booleans, defined by a *key* followed by `:` and a *value*, i.e.:

```
model: fesom
version: "2.0"
time_step: 1800
```

- **Lists**: a collection of elements defined by a *key* followed by `:` and an indented list of *elements* (numbers, strings or booleans) starting with −, i.e.:

```
namelists:
        - namelist.config
        - namelist.forcing
        - namelist.oce
```

  or a list of the same *elements* separated by `,` inside square brackets `[elem1, elem2]`:

```
namelists: [namelist.config, namelist.forcing, namelist.oce]
```

- **Dictionaries**: a collection of *scalars*, *lists* or *dictionaries* nested inside a general *key*, i.e.:

```
config_files:
        config:  config
        forcing: forcing
        ice:     ice
```

Some relevant **properties** of the YAML format are:

- Only **white spaces** can be used for indentation. **Tabs are not allowed**.
- Indentation can be used to structure information in as many levels as required, i.e. a dictionary choose_resolution that contains a list of dictionaries (T63, T31 and T127):

```
choose_resolution:
        T63:
                levels: "L47"
                time_step: 450
                [ ... ]
        T31:
                levels: "L19"
                time_step: 450
                [ ... ]
        T127:
                levels: "L47"
                time_step: 200
                [ ... ]
```

- This data can be easily imported as *Python* dictionaries, which is part of what the *esm_parser* does.
- `:` should always be **followed** by a *white space*.
- **Strings** can be written both **inside quotes** (`key:  "string"` or `key:  'string'`) **or unquoted** (`key: string`).
- *YAML* format is **case sensitive**.
- It is possible to add **comments** to YAML files using # before the comment (same as in Python).

6.2 ESM-Tools Extended YAML Syntax

Warning: Work in progress. This chapter might be incomplete. Red statements might be imprecise or not true.

ESM-Tools offers extended functionality of the *YAML* files through the *esm_parser*. The following subsections list the extended *ESM-Tools* syntax for *YAML* files including calendar and math operations (see *Math and Calendar Operations*). The yaml:YAML Elements section lists the *YAML* elements needed for configuration files and runscripts.

6.2.1 Variable Calls

Variables defined in a *YAML* file can be invoked on the same file or in oder files provided that the file where it is defined is read for the given operation. The syntax for calling an already defined variable is:

```
"${name_of_the_variable}"
```

Variables can be nested in sections. To define a variable using the value of another one that is nested on a section the following syntax is needed:

```
"${<section>.<variable>}"
```

When using *esm_parser*, variables in components, setups, machine files, general information, etc., are grouped under sections of respective names (i.e. `general`, `ollie`, `fesom`, `awicm`, ...). To access a variable from a different file than the one in which it is declared it is necessary to reference the file name or label as it follows:

```
"${<file_label>.<section>.<variable>}"
```

Example

Lets take as an example the variable `ini_parent_exp_id` inside the `general` section in the *FESOM-REcoM* runscript `runscripts/fesom-recom/fesom-recom-ollie-restart-daily.yaml`:

```
general:
        setup_name: fesom-recom
        [ ... ]
        ini_parent_exp_id: restart_test
        ini_restart_dir: /work/ollie/mandresm/esm_yaml_test/${ini_parent_exp_id}/
↪restart/
        [ ... ]
```

Here we use `ini_parent_exp_id` to define part of the restart path `ini_restart_dir`. `general.ini_restart_dir` is going to be called from the *FESOM-REcoM* configuration file `configs/setups/fesom-recom/fesom-recom.yaml` to define the restart directory for *FESOM* `fesom.ini_restart_dir`:

```
[ ... ]
ini_restart_dir: "${general.ini_restart_dir}/fesom/"
[ ... ]
```

Note that this line adds the subfolder `/fesom/` to the subdirectory.

If we would like to invoke from the same runscript some of the variables defined in another file, for example the `useMPI` variable in `configs/machines/ollie.yaml`, then we would need to use:

```
a_new_variable: "${ollie.useMPI}"
```

Bare in mind that these examples will only work if both *FESOM* and *REcoM* are involved in the *ESM-Tool* task triggered and if the task is run in *Ollie* (i.e. it will work for `esm_runscripts fesom-recom-ollie-restart-daily.yaml -e <experiment_id> ...`).

6.2.2 Switches (choose_)

A *YAML* list named as choose_<variable> function as a *switch* that evaluates the given variable. The nested element *keys* inside the choose_<variable> act as *cases* for the switch and the *values* of this elements are only defined outside of the choose_<variable> if they belong to the selected case_key:

```
variable_1: case_key_2

choose_variable_1:
        case_key_1:
                configuration_1: value
                configuration_2: value
                [ ... ]
        case_key_2:
                configuration_1: value
                configuration_2: value
                [ ... ]
        "*":
                configuration_1: value
                configuration_2: value
                [ ... ]
```

The key "*" or * works as an *else*.

Example

An example that can better illustrate this general description is the *FESOM 2.0* resolution configuration in <PATH>/esm_tools/configs/fesom/fesom-2.0.yaml:

```
resolution: CORE2

choose_resolution:
        CORE2:
                nx: 126858
                mesh_dir: "${pool_dir}/meshes/mesh_CORE2_final/"
                nproc: 288
        GLOB:
                nx: 830305
```

Here we are selecting the CORE2 as default configuration set for the resolution variable, but we could choose the GLOB configuration in another *YAML* file (i.e. a runscript), to override this default choice.

In the case in which resolution: CORE2, then nx, mesh_dir and nproc will take the values defined inside the choose_resolution for CORE2 (126858, runscripts/fesom-recom/fesom-recom-ollie-restart-daily.yaml, and 288 respectively), once resolved by the *esm_parser*, at the same **nesting level** of the choose_resolution.

Note: choose_versions inside configuration files is treated in a special way by the *esm_master*. To avoid conflicts in case an additional choose_versions is needed, include the compilation information inside a compile_infos section (including the choose_versions switch containning compilation information). Outside of this exception, it is possible to use as many choose_<variable> repetitions as needed.

6.2.3 Append to an Existing List (add_)

Given an existing list list1 or dictionary:

```
list1:
    - element1
    - element2
```

it is possible to add members to this list/dictionary by using the following syntax:

```
add_list1:
    - element3
    - element4
```

so that the variable list1 at the end of the parsing will contain [element1, element2, element3, element4]. This is not only useful when you need to build the list piecewise (i.e. and expansion of a list inside a choose_ switch) but also as the *YAML File Hierarchy* will cause repeated variables to be overwritten. Adding a nested dictionary in this way merges the add_<dictionary> content into the <dictionary> with priority to add_<dictionary> elements inside the same file, and following the *YAML File Hierarchy* for different files.

Properties

- It is possible to have multiple add_ for the same variable in the same or even in different files. That means that all the elements contained in the multiple add_ will be added to the list after the parsing.

Exceptions

Exceptions to add_ apply only to the environment and namelist _changes (see *Environment and Namelist Changes (_changes)*). For variables of the type _changes, an add_ is only needed if the same _changes block repeats inside the same file. Otherwise, the _changes block does not overwrite the same _changes block in other files, but their elements are combined.

Example

In the configuration file for *ECHAM* (configs/components/echam/echam.yaml) the list input_files is declared as:

```
[ ... ]

input_files:
    "cldoptprops": "cldoptprops"
    "janspec": "janspec"
    "jansurf": "jansurf"
    "rrtmglw": "rrtmglw"
    "rrtmgsw": "rrtmgsw"
    "tslclim": "tslclim"
    "vgratclim": "vgratclim"
    "vltclim": "vltclim"

[ ... ]
```

However different *ECHAM* scenarios require additional input files, for example the HIST scenario needs a MAC-SP element to be added and we use the add_ functionality to do that:

```
[ ... ]
choose_scenario:
    [ ... ]
    HIST:
        forcing_files:
```

(continues on next page)

(continued from previous page)

```
            [ ... ]
        add_input_files:
            MAC-SP: MAC-SP
    [ ... ]
```

An example for the `_changes` **exception** can be also found in the same `ECHAM` configuration file. Namelist changes necessary for *ECHAM* are defined inside this file as:

```
[ ... ]

namelist_changes:
    namelist.echam:
        runctl:
            out_expname: ${general.expid}
            dt_start:
                - ${pseudo_start_date!year}
                - ${pseudo_start_date!month}
                [ ... ]
```

This changes specified here will be combined with changes in other files (i.e. `echam.namelist_changes` in the coupled setups *AWICM* or *AWIESM* configuration files), not overwritten. However, *ECHAM*'s version `6.3.05p2-concurrent_radiation` needs of further namelist changes written down in the same file inside a `choose_` block and for that we need to use the `add_` functionality:

```
[ ... ]

choose_version:
    [ ... ]
    6.3.05p2-concurrent_radiation:
        [ ... ]
        add_namelist_changes:
            namelist.echam:
                runctl:
                    npromar: "${npromar}"
                parctl:

[ ... ]
```

6.2.4 Remove Elements from a List/Dictionary (`remove_`)

It is possible to remove elements inside list or dictionaries by using the `remove_` functionality which syntax is:

```
remove_<dictionary>: [<element_to_remove1>, <element_to_remove2>, ... ]
```

or:

```
remove_<dictionary>:
        - <element_to_remove1>
        - <element_to_remove2>
        -  ...
```

You can also remove specific nested elements of a dictionary separating the *keys* for the path by `.`:

```
remove_<model>.<dictionary>.<subkey1>.<subkey2>: [<element_to_remove1>, <element_to_
↪remove2>, ... ]
```

6.2.5 Math and Calendar Operations

The following math and calendar operations are supported in *YAML* files:

Arithmetic Operations

An element of a *YAML* file can be defined as the result of the addition, subtraction, multiplication or division of variables with the format:

```
key: "$(( ${variable_1} operator ${variable_2} operator ... ${variable_n} ))"
```

The *esm_parser* supports calendar operations through *esm_calendar*. When performing calendar operations, variables that are not given in date format need to be followed by their `unit` for the resulting variable to be also in date format, i.e.:

```
runtime: $(( ${end_date} - ${time_step}seconds ))
```

`time_step` is a variable that is not given in date format, therefore, it is necessary to use `seconds` for `runtime` to be in date format. Another example is to subtract one day from the variable `end_date`:

```
$(( ${end_date} - 1days ))
```

The units available are:

Units supported by arithmetic operations	
calendar units	seconds minutes days months years

Extraction of Date Components from a Date

It is possible to extract date components from a *date variable*. The syntax for such an operation is:

```
"${variable!date_component}"
```

An example to extract the year from the `initial_time` variable:

```
yearnew: "${initial_date!syear}"
```

If `initial_date` was 2001-01-01T00:00:00, then `yearnew` would be 2001.

The date components available are:

Date components	
ssecond	Second from a given date.
sminute	Minute from a given date.
shour	Hour from a given date.
sday	Day from a given date.
smonth	Month from a given date.
syear	Year from a given date.
sdoy	Day of the year, counting from Jan. 1.

6.2.6 Globbing

Globbing allows to use * as a wildcard in filenames for restart, input and output files. With this feature files can be copied from/to the work directory whose filenames are not completely known. The syntax needed is:

```
file_list: common_pathname*common_pathname
```

Note that this also works together with the *List Loops*.

Example

The component *NEMO* produces one restart file per processor, and the part of the file name relative to the processor is not known. In order to handle copying of restart files under this circumstances, globbing is used in *NEMO*'s configuration file (`configs/components/nemo/nemo.yaml`):

```
[ ... ]

restart_in_sources:
    restart_in: ${expid}_${prevstep_formatted}_restart*_${start_date_m1!syear!smonth!
↪sday}_*.nc
restart_out_sources:
    restart_out: ${expid}_${newstep_formatted}_restart*_${end_date_m1!syear!smonth!
↪sday}_*.nc

[ ... ]
```

This will include inside the `restart_in_sources` and `restart_out_sources` lists, all the files sharing the specified common name around the position of the * symbol, following the same rules used by the Unix shell.

6.2.7 Environment and Namelist Changes (_changes)

The functionality _changes is used to control environment, namelist and coupling changes. This functionality can be used from config files, but also runscripts. If the same type of _changes is used both in config files and a runscript for a simulation, the dictionaries are merged following the hierarchy specified in the *YAML File Hierarchy* chapter.

Environment Changes

Environment changes are used to make changes to the default environment defined in the machine files (esm_tools/configs/machines/<name_of_the_machine>.yaml). There are three types of environment changes:

Key	Description
environment_changes	Changes for both the compilation and the runtime environments.
compiletime_environment_changes	Changes to the environment applied only during compilation.
runtime_environment_changes	Changes to the environment applied only during runtime.

Two types of *yaml* elements can be nested inside an environment changes: add_module_actions and add_export_vars.

- Use add_module_actions to include one *module* command or a list of them. The shell command module is already invoked by *ESM-Tools*, therefore you only need to list the options (i.e. load/unload <module_name>).
- Use add_export_vars to export one or a list of environment variables. Shell command export is not needed here, just define the variable as VAR_NAME=VAR_VALUE or as a nested dictionary.

Example

fesom.yaml

The model *FESOM* needs some environment changes for compiling in *Mistral* and *Blogin* HPCs, which are included in *FESOM*'s configuration file (esm_tools/configs/components/fesom/fesom.yaml):

```
[ ... ]

compiletime_environment_changes:
        add_export_vars:
                takenfrom:      fesom1
choose_computer.name:
        mistral:
                add_compiletime_environment_changes:
                        add_module_actions:
                                - "unload gcc"
                                - "load gcc/4.8.2"
        blogin:
                add_compiletime_environment_changes:
                        add_export_vars:
                                - "NETCDF_DIR=/sw/dataformats/netcdf/intel.18/4.7.3/
↪skl/"
                                - "LD_LIBRARY_PATH=$NETCDF_DIR/lib/:$LD_LIBRARY_PATH"
                                - "NETCDF_CXX_INCLUDE_DIRECTORIES=$NETCDF_DIR/include"
                                - "NETCDF_CXX_LIBRARIES=$NETCDF_DIR/lib"
                                - "takenfrom='fesom1'"

runtime_environment_changes:
        add_export_vars:
```

(continues on next page)

(continued from previous page)

```
                AWI_FESOM_YAML:
                        output_schedules:
                                -
                                        vars: [restart]
                                        unit: ${restart_unit}
                                        first: ${restart_first}
                                        rate: ${restart_rate}
                                -
                                        [ ... ]
```

Independently of the computer, `fesom.yaml` exports always the `takenfrom` variable for compiling. Because `compiletime_environment_changes` is already defined for that purpose, any `compiletime_environment_changes` in a `choose_` block needs to have an `add_` at the beginning. Here we see that a `choose_` block is used to select which changes to apply compile environment (`add_compiletime_environment_changes`) depending on the HPC system we are in (*Mistral* or *Blogin*). For more details on how to use the `choose_` and `add_` functionalities see *Switches (choose_)* and *Append to an Existing List (add_)*.

We also see here how `runtime_environment_changes` is used to add nested information about the output schedules for *FESOM* into an `AWI_FESOM_YAML` variable that will be exported to the runtime environment.

Changing Namelists

It is also possible to specify namelist changes to a particular section of a namelist:

```
echam:
        namelist_changes:
                namelist.echam:
                        runctl:
                                l_orbvsop87: false
                        radctl:
                                co2vmr: 217e-6
                                ch4vmr: 540e-9
                                n2ovmr: 245e-9
                                cecc: 0.017
                                cobld: 23.8
                                clonp: -0.008
                                yr_perp: "remove_from_namelist"
```

In the example above, the *namelist.echam* file is changed in two specific chapters, first the section `runctrl` parameter `l_orbsvop87` is set to `false`, and appropriate gas values and orbital values are set in `radctl`. Note that the special entry `"remove_from_namelist` is used to delete entries. This would translate the following fortran namelist (trucated):

```
&runctl
    l_orbvsop87 = .false.
/

&radctl
    co2vmr = 0.000217
    ch4vmr = 5.4e-07
    n2ovmr = 2.45e-07
    cecc = 0.017
    cobld = 23.8
```

(continues on next page)

(continued from previous page)

```
    clonp = -0.008
/
```

Note that, although we set `l_orbsvop87` to be `false`, it is translated to the namelist as a fortran boolean (`.false.`). This occurs because *ESM-Tools* "understands" that it is writing a fortan namelist and transforms the *yaml* booleans into fortran.

For more examples, check the recipe in the cookbook (*Changing Namelist Entries from the Runscript*).

Coupling changes

Coupling changes (`coupling_changes`) are typically invoked in the coupling files (`esm_tools/configs/couplings/`), executed before compilation of coupled setups, and consist of a list of shell commands to modify the configuration and make files of the components for their correct compilation for coupling.

For example, in the `fesom-1.4+echam-6.3.04p1.yaml` used in *AWICM-1.0*, `coupling_changes` lists two `sed` commands to apply the necessary changes to the `CMakeLists.txt` files for both *FESOM* and *ECHAM*:

```
components:
- echam-6.3.04p1
- fesom-1.4
- oasis3mct-2.8
coupling_changes:
- sed -i '/FESOM_COUPLED/s/OFF/ON/g' fesom-1.4/CMakeLists.txt
- sed -i '/ECHAM6_COUPLED/s/OFF/ON/g' echam-6.3.04p1/CMakeLists.txt
```

6.2.8 List Loops

This functionality allows for basic looping through a *YAML list*. The syntax for this is:

```
"[[list_to_loop_through-->ELEMENT_OF_THE_LIST]]"
```

where `ELEMENT_OF_THE_LIST` can be used in the same line as a variable. This is particularly useful to handle files which names contain common strings (i.e. *outdata* and *restart* files, see *File Dictionaries*).

The following example uses the list loop functionality inside the `fesom-2.0.yaml` configuration file to specify which files need to be copied from the *work* directory of runs into the general experiment *outdata* directory. The files to be copied for runs modeling a couple of months in year 2001 are `a_ice.fesom.2001.nc`, `alpha.fesom.2001.nc`, `atmice_x.fesom.2001.nc`, etc. The string `.fesom.2001.nc` is present in all files so we can use the list loop functionality together with calendar operations (*Math and Calendar Operations*) to have a cleaner and more generalized configure file. First, you need to declare the list of unshared names:

```
outputs: [a_ice,alpha,atmice_x, ... ]
```

Then, you need to declare the `outdata_sources` dictionary:

```
outdata_sources:
      "[[outputs-->OUTPUT]]": OUTPUT.fesom.${start_date!syear}.nc
```

Here, `"[[outputs-->OUTPUT]]":` provides the *keys* for this dictionary as `a_ice`, `alpha`, `atmice_x`, etc., and `OUTPUT` is later used in the *value* to construct the complete file name (`a_ice.fesom.2001.nc`, `alpha.fesom.2001.nc`, `atmice_x.fesom.2001.nc`, etc.).

Finally, `outdata_targets` dictionary can be defined to give different names to *outdata* files from different runs using *calendar operations*:

```
outdata_targets:
      "[[outputs-->OUTPUT]]": OUTPUT.fesom.${start_date!syear!smonth}.${start_date!
↪sday}.nc
```

The values for the *keys* `a_ice`, `alpha`, `atmice_x`, ..., will be `a_ice.fesom.200101.01.nc`, `alpha.fesom.200101.01.nc`, `atmice_x.fesom.200101.01.nc`, ..., for a January run, and `a_ice.fesom.200102.01.nc`, `alpha.fesom.200102.01.nc`, `atmice_x.fesom.200102.01.nc`, ..., for a February run.

6.2.9 File Dictionaries

File dictionaries are a special type of *YAML* elements that are useful to handle input, output, forcing, logging, binary and restart files among others (see *File dictionary types* table), and that are normally defined inside the *configuration files* of models. File dictionary's *keys* are composed by a file dictionary `type` followed by _ and an `option`, and the *elements* consist of a list of `file_tags` as *keys* with their respective `file_paths` as *values*:

```
type_option:
      file_tag1: file_path1
      file_tag2: file_path2
```

The `file_tags` need to be consistent throughout the different `options` for files to be correctly handled by ESM-Tools. Exceptionally, `sources` files can be tagged differently but then the option `files` is required to link sources tags to general tags used by the other options (see *File dictionary options* table below).

File dictionary types

Key	Description
analysis	User's files for their own analysis tools (i.e. to be used in the pre-/postprocessing).
bin	Binary files.
config	Configure sources.
couple	Coupling files.
ignore	Files to be ignored in the copying process.
forcing	Forcing files. An example is described at the end of this section.
log	Log files.
mon	Monitoring files.
outdata	Output configuration files. A concise example is described in *List Loops*.
restart_in	Restart files to be copied from the **experiment directory** into the **run directory** (see *Experiment Directory Structure*), during the beginning of the *computing phase* (e.g. to copy restart files from the previous step into the new run folder).
restart_out	Restart files to be copied from the **run directory** into the **experiment directory** (see *Experiment Directory Structure*), during the *tidy and resubmit phase* (e.g. to copy the output restart files from a finished run into the **experiment directory** for later use the next run).
viz	Files for the visualization tool.

File dictionary options

Key	Description
sources	Source file paths or source file names to be copied to the target path. **Without this option no files will be handled by ESM-Tools**. If `targets` option is not defined, the files are copied into the default *target* directory with the same name as in the *source* directory. In that case, if two files have the same name they are both renamed to end in the dates corresponding to their run (`file_name.extension_YYYYMMDD_YYYYMMDD`).
files	Links the general file tags (*key*) to the *source* elements defined in `sources`. `files` **is optional**. If not present, all *source* files are copied to the *target* directory, and the *source tags* need to be the same as the ones in `in_work` and `targets`. If present, only the *source* files included in `files` will be copied (see the *ECHAM* forcing files example below).
in_work	Files inside the *work* directory of a run (`<base_dir>/<experiment_name>/run_date1_date2/work`) to be transferred to the *target* directory. This files copy to the *target* path even if they are not included inside the `files` option. `in_work` **is optional**.
targets	Paths and new names to be given to files transferred from the *sources* directory to the *target* directory. A concised example is described in *List Loops*. `targets` **is optional**.

File paths can be absolute, but most of the `type_option` combinations have a default folder assigned, so that you can choose to specify only the file name. The default folders are:

Default folders	sources	in_work	targets
bin			
config			
ignore			
forcing			
log			
outdata	`<base_dir>/<experiment_name>/run_date1_date2/work`	`<base_dir>/<experiment_name>/run_date1_date2/work`	`<base_dir>/<experiment_name>/outdata/<model>`
restart_in			
restart_out			

Example for ECHAM forcing files

The *ECHAM* configuration file (`<PATH>/configs/echam/echam.yaml`) allows for choosing different scenarios for a run. These scenarios depend on different combinations of forcing files. File sources for all cases are first stored in `echam.datasets.yaml` (a `further_reading` file) as:

```
forcing_sources:
      # sst
      "amipsst":
              "${forcing_dir}/amip/${resolution}_amipsst_@YEAR@.nc":
                      from: 1870
                      to: 2016
      "pisst": "${forcing_dir}/${resolution}${ocean_resolution}_piControl-LR_sst_1880-
↪2379.nc"

      # sic
      "amipsic":
              "${forcing_dir}/amip/${resolution}_amipsic_@YEAR@.nc":
                      from: 1870
                      to: 2016
      "pisic": "${forcing_dir}/${resolution}${ocean_resolution}_piControl-LR_sic_1880-
↪2379.nc"
```

(continues on next page)

(continued from previous page)

```
      [ ... ]
```

Here `forcing_sources` store **all the sources** necessary for all *ECHAM* scenarios, and tag them with source *keys* (`amipsst`, `pisst`, ...). Then, it is possible to choose among these source files inside the scenarios defined in `echam.yaml` using `forcing_files`:

```
choose_scenario:
      "PI-CTRL":
              forcing_files:
                      sst: pisst
                      sic: pisic
                      aerocoarse: piaerocoarse
                      aerofin: piaerofin
                      aerofarir: piaerofarir
                      ozone: piozone
      PALEO:
              forcing_files:
                      aerocoarse: piaerocoarse
                      aerofin: piaerofin
                      aerofarir: piaerofarir
                      ozone: piozone
      [ ... ]
```

This means that for a scenario `PI-CTRL` the files that are handled by ESM-Tools will be **exclusively** the ones specified inside `forcing_files`, defined in the `forcing_sources` as `pisst`, `pisic`, `piaerocoarse`, `piaerofin`, `piaerofarir` and `piozone`, and they are tagged with new general *keys* (`sst`, `sic`, ...) that are common to all scenarios. The source files not included in `forcing_files` won't be used.

CHAPTER

SEVEN

YAML FILE HIERARCHY

7.1 Hierarchy of YAML configuration files

The following graph illustrates the hierarchy of the different YAML configuration files.

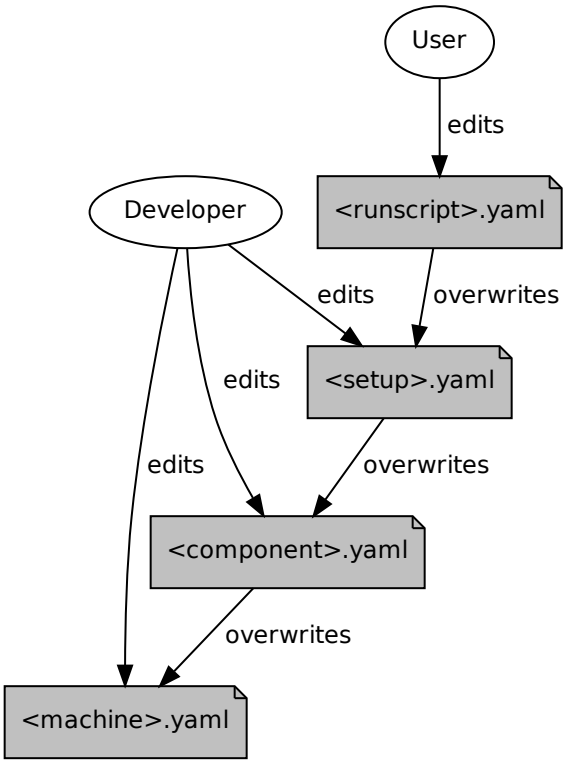

Fig. 1: ESM-Tools configuration files hierarchy

CHAPTER

EIGHT

ESM-TOOLS VARIABLES

The *esm_parser* is used to read the multiple types of *YAML* files contained in *ESM-Tools* (i.e. model and coupling configuration files, machine configurations, runscripts, etc.). Each of these *YAML* files can contain two type of *YAML* elements/variables:

- **Tool-specific elements**: *YAML-scalars*, *lists* or *dictionaries* that include instructions and information used by *ESM-Tools*. These elements are predefined inside the *esm_parser* or other packages inside *ESM-Tools* and are used to control the *ESM-Tools* functionality.
- **Setup/model elements**: *YAML-scalars*, *lists* of *dictionaries* that contain information defined in the model/setup config files (i.e. `awicm.yaml`, `fesom.yaml`, etc.). This information is model/setup-specific and causes no effect unless it is combined with the **tool-specific elements**. For example, in `fesom.yaml` for *FESOM-1.0* the variable `asforcing` exists, however this means nothing to *ESM-Tools* by its own. In this case, this variable is used in `namelist_changes` (a tool-specific element) to state the type of forcing to be used and this is what actually makes a difference to the simulation. The advantage of having this variable already defined and called in `namelist_changes`, in the `fesom.yaml` is that the front-end user can simply change the forcing type by changing the value of `asforcing` (no need for the front-end user to use `namelist_changes`).

The following subsection lists and describes the **Tool-specific elements** used to operate *ESM-Tools*.

Note: Most of the **Tool-specific elements** can be defined in any file (i.e. *configuration file*, *runscript*, ...) and, if present in two files used by ESM-Tools at a time, the value is chosen depending on the ESM-Tools file priority/read order (*YAML File Hierarchy*). Ideally, you would like to declare as many elements as possible inside the *configuration files*, to be used by default, and change them in the *runscripts* when necessary. However, it is ultimately up to the user where to setup the Tool-specific elements.

8.1 Tool-Specific Elements/Variables

The following keys should/can be provided inside configuration files for models (`<PATH>/esm_tools/configs/components/<name>/<name>.yaml`), coupled setups (`<PATH>/esm_tools/configs/setups/<name>/<name>.yaml`) and runscripts. You can find runscript templates in `esm_tools/runscripts/templates/`).

8.1.1 Installation variables

Key	Description
model	Name of the model/setup as listed in the config files (`esm_tools/configs/components` for models and `esm_tools/configs/setups` for setups).
setup_name	Name of the coupled setup.
version	Version of the model/setup (one of the available options in the `available_versions` list).
available_versions	List of supported versions of the component or coupled setup.
git-repository	Address of the model's git repository.
branch	Branch from where to clone.
destination	Name of the folder where the model is downloaded and compiled, in a coupled setup.
comp_command	Command used to compile the component.
install_bins	Path inside the component folder, where the component is compiled by default. This path is necessary because, after compilation, ESM-Tools needs to copy the binary from this path to the `<component/setup_path>/bin` folder.

8.1.2 Runtime variables

Key	Description
account	User account of the HPC system to be used to run the experiment.
model_dir	Absolute path of the model directory (where it was installed by *esm_master*).
setup_dir	Absolute path of the setup directory (where it was installed by *esm_master*).
executable	Name of the component executable file, as it shows in the `<component/setup_path>/bin` after compilation.
compute_time	Estimated computing time for a run, used for submitting a job with the job scheduler.
time_step	Time step of the component in seconds.
lresume	Boolean to indicate whether the run is an initial run or a restart.
pool_dir	Path to the pool directory to read in mesh data, forcing files, inputs, etc.
namelists	List of namelist files required for the model.
namelist_changes	Functionality to handle changes in the namelists from the yaml files (see *Changing Namelists*).
nproc	Number of processors to use for the model.
nproca/nprocb	Number of processors for different MPI tasks/ranks. Incompatible with `nproc`.
base_dir	Path to the directory that will contain the experiment folder (where the experiment will be run and data will be stored).
post_processing	Boolean to indicate whether to run postprocessing or not.
File Dictionaries	*YAML* dictionaries used to handle input, output, forcing, logging, binary and restart files (see *File Dictionaries*).
expid	ID of the experiment. This variable can also be defined when calling `esm_runscripts` with the `-e` flag.
ini_restart_exp_id	ID of the restarted experiment in case the current experiment has a different `expid`. For this variable to have an effect `lresume` needs to be `true` (e.g. the experiment is a restart).
ini_restart_dir	Path of the restarted experiment in case the current experiment runs in a different directory. For this variable to have an effect `lresume` needs to be `true` (e.g. the experiment is a restart).
execution_command	Command for executing the component, including `${executable}` and the necessary flags.

8.1.3 Calendar variables

Key	Description
initial_date	Date of the beginning of the **simulation** in the format YYYY-MM-DD. If the simulation is a restart, `initial_date` marks the beginning of the restart.
final_date	Date of the end of the **simulation** in the format YYYY-MM-DD.
start_date	Date of the beginning of the **current run**.
end_date	Date of the end of the **current run**.
current_date	Current date of the run.
next_date	Next run initial date.
nyear, nmonth, nday, nhour, nminute	Number of time unit per run. They can be combined (i.e. `nyear: 1` and `nmonth: 2` implies that each run will be 1 year and 2 months long).
parent_date	Ending date of the previous run.

8.1.4 Coupling variables

Key	Description
grids	List of grids and their parameters (i.e. `name`, `nx`, `ny`, etc.).
coupling_fields	List of coupling field dictionaries containing coupling field variables.
nx	When using *oasis3mct*, used inside `grids` to define the first dimension of the grid.
ny	When using *oasis3mct*, used inside `grids` to define the second dimension of the grid.
coupling_methods	List of coupling methods and their parameters (i.e. `time_transformation`, `remapping`, etc.).
time_transformation	Time transformation used by *oasis3mct*, defined inside `coupling_methods`.
remapping	Remappings and their parameters, used by *oasis3mct*, defined inside `coupling_methods`.

8.1.5 Other variables

Key	Description
metadata	List to incude descriptive information about the model (i.e. `Authors`, `Institute`, `Publications`, etc.) used to produce the content of *Supported Models*. This information should be organized in nested *keys* followed by the corresponding description. Nested *keys* do not receive a special treatment meaning that you can include here any kind of information about the model. Only the *Publications key* is treated in a particular way: it can consist of a single element or a *list*, in which each element contains a link to the publication inside <> (i.e. `- Title, Authors, Journal, Year. <https://doi.org/...>`).

CHAPTER

NINE

SUPPORTED MODELS

9.1 AMIP

9.2 DEBM

Institute	AWI
Description	dEBM is a surface melt scheme to couple ice and climate models in paleo applications.
Publications	Krebs-Kanzow, U., Gierz, P., and Lohmann, G., Brief communication: An Ice surface melt scheme including the diurnal cycle of solar radiation, The Cryosphere Discuss., accepted for publication
License	MIT

9.3 ECHAM

Institute	MPI-Met
Description	The ECHAM atmosphere model, major version 6
Authors	Bjorn Stevens (bjorn.stevens@mpimet.mpg.de) among others at MPI-Met
Publications	Atmosphericcomponent of the MPI-M earth system model: ECHAM6
License	Please make sure you have a license to use ECHAM. In case you are unsure, please contact redmine. . .

9.4 ESM_INTERFACE

Institute	Alfred Wegener Institute
Description	Coupling interface for a modular coupling approach of ESMs.
Authors	Nadine Wieters (nadine.wieters@awi.de)
Publications	`None`_
License	None

9.5 FESOM

Institute	Alfred Wegener Institute
Description	Multiresolution sea ice-ocean model that solves the equations of motion on unestructured meshes
Authors	Dmitry Sidorenko (Dmitry.Sidorenko@awi.de), Nikolay V. Koldunov (nikolay.koldunov@awi.de)
Publications	The Finite-volumE Sea ice-Ocean Model (FESOM2) Scalability and some optimization of the Finite-volumE Sea ice-Ocean Model, Version 2.0 (FESOM2)
License	Please make sure you have a licence to use FESOM. In case you are unsure, please contact redmine...

9.6 FESOM_MESH_PART

Description	The FESOM Mesh Partioner (METIS)

9.7 HDMODEL

9.8 ICON

Institute	MPI-Met
Description	The ICON atmosphere model, major version 2
Authors	Marco Giorgetta (marco.giorgetta@mpimet.mpg.de), Peter Korn, Christian Reick, Reinhard Budich
Publications	ICON-A, the Atmosphere Component of the ICON Earth System Model: I. Model Description
License	Please make sure you have a license to use ICON. In case you are unsure, please contact redmine...

9.9 JSBACH

9.10 MPIOM

Institute	MPI-Met
Description	The ocean-sea ice component of the MPI-ESM. MPIOM is a primitive equation model (C-Grid, z-coordinates, free surface) with the hydrostatic and Boussinesq assumptions made.
Authors	Till Maier-Reimer, Helmuth Haak, Johann Jungclaus
Publications	Characteristics of the ocean simulations in the Max Planck Institute Ocean Model (MPIOM) the ocean component of the MPI-Earth system model The Max-Planck-Institute global ocean/sea ice model with orthogonal curvilinear coordinates
License	Please make sure you have a licence to use MPIOM. In case you are unsure, please contact redmine...

9.11 NEMO

Organization	Nucleus for European Modelling of the Ocean
Institute	IPSL
Description	NEMO standing for Nucleus for European Modelling of the Ocean is a state-of-the-art modelling framework for research activities and forecasting services in ocean and climate sciences, developed in a sustainable way by a European consortium.
Authors	Gurvan Madec and NEMO System Team (nemo_st@locean-ipsl.umpc.fr)
Publications	NEMO ocean engine
License	Please make sure you have a license to use NEMO. In case you are unsure, please contact redmine...

9.12 NEMOBASEMODEL

9.13 OASIS3MCT

9.14 OIFS

Institute	ECMWF
Description	OpenIFS provides research institutions with an easy-to-use version of the ECMWF IFS (Integrated Forecasting System).
Authors	Glenn Carver (openifs-support@ecmwf.int)
Website	https://www.ecmwf.int/en/research/projects/openifs
License	Please make sure you have a licence to use OIFS. In case you are unsure, please contact redmine...

9.15 PISM

Institute	UAF and PIK
Description	The Parallel Ice Sheet Model (PISM) is an open source, parallel, high-resolution ice sheet model.
Authors	Ed Bueler, Jed Brown, Anders Levermann, Ricarda Winkelmann and many more (uaf-pism@alaska.edu)
Publications	Shallow shelf approximation as a "sliding law" in a thermomechanically coupled ice sheet model The Potsdam parallel ice sheet model (PISM-PIK) - Part 1: Model description
License	GPL 3.0

9.16 RECOM

Institute	AWI
Description	REcoM (Regulated Ecosystem Model) is an ecosystem and biogeochemistry model.
Authors	Judith Hauck, Ozgur Gurses
Publications	Seasonally different carbon flux changes in the Southern Ocean in response to the southern annular mode Arctic Ocean biogeochemistry in the high resolution FESOM 1.4-REcoM2 model
License	Please make sure you have a licence to use REcoM. In case you are unsure, please contact redmine...

9.17 RNFMAP

9.18 SAMPLE

9.19 SCOPE

Institute	Alfred Wegener Institute
Description	The Script-Based Coupler
Authors	Paul Gierz (pgierz@awi.de)

9.20 VILMA

9.21 XIOS

Institute	IPSL and CEA
Description	A library dedicated to I/O management in climate codes.
Authors	Yann Meurdesoif (yann.meurdesoif@cea.fr)
Website	https://portal.enes.org/models/software-tools/xios
License	Please make sure you have a licence to use XIOS. In case you are unsure, please contact redmine...

9.22 YAC

Information	For more information about YAC please go to the webpage: https://dkrz-sw.gitlab-pages.dkrz.de/yac/index.html

CHAPTER

TEN

ESM MASTER

10.1 Usage: esm_master

To use the command line tool `esm_master`, just enter at a prompt:

```
$ esm_master
```

The tool may ask you to configure your settings; which are stored in your home folder under `${HOME}/.esmtoolsrc`. A list of avaiable models, coupled setups, and available operations are printed to the screen, e.g.:

```
setups:
   awicm:
      1.0: ['comp', 'clean', 'get', 'update', 'status', 'log', 'install', 'recomp']
      CMIP6: ['comp', 'clean', 'get', 'update', 'status', 'log', 'install', 'recomp']
      2.0: ['comp', 'clean', 'get', 'update', 'status', 'log', 'install', 'recomp']
[...]
```

As can be seen in this example, `esm_master` supports operations on the coupled setup awicm in the versions 1.0, CMIP6 and 2.0; and what the tool can do with that setup. You execute `esm_master` by calling:

```
$ esm_master operation-software-version,
```

e.g.:

```
$ esm_master install-awicm-2.0
```

By default, `esm_master` supports the following operations:

get: Cloning the software from a repository, currently supporting git and svn

conf: Configure the software (only needed by mpiesm and icon at the moment)

comp: Compile the software. If the software includes libraries, these are compiled first. After compiling the binaries can be found in the subfolders `bin` and `lib`.

clean: Remove all the compiled object files.

install: Shortcut to get, then conf, then comp.

recomp: Shortcut to conf, then clean, then comp.

update: Get the newest commit of the software from the repository.

status: Get the state of the local database of the software (e.g. `git status`)

log: Get a list of the last commits of the local database of the software (e.g. `git log`)

To download, compile, and install `awicm-2.0`; you can say:

```
$ esm_master install-awicm-2.0
```

This will trigger a download, if needed a configuration, and a compilation process. Similarly, you can recompile with `recomp-XXX`, clean with `clean-XXX`, or do individual steps, e.g. `get`, `configure`, `comp`.

The download and installation will always occur in the **current working directory**.

You can get further help with:

```
$ esm_master --help
```

10.2 Configuring esm-master for Compile-Time Overrides

It is possible that some models have special compile-time settings that need to be included, overriding the machine defaults. Rather than placing these changes in `configs/machines/NAME.yaml`, they can be instead placed in special blocks of the component or model configurations, e.g.:

```
compiletime_environment_changes:
        add_export_vars:
                [ ... ]
```

The same is also possible for specifying `runtime_environment_changes`.

CHAPTER

ELEVEN

ESM-VERSIONS

New with the Tools version 3.1.5, you will find an executable in your path called `esm_version`. This was added by Paul Gierz to help the user / developer to keep track of / upgrade the python packages belonging to ESM Tools.

11.1 Usage

It doesn't matter from which folder you call `esm_versions`. You have two subcommands:

```
esm_versions check                      gives you the version number of each
                                        installed esm python package
esm_versions upgrade                    upgrades all installed esm python
                                        packages to the newest version
                                        of the release branch
```

Notice that you can also upgrade single python packages, e.g.:

```
esm_versions upgrade esm_parser         upgrades only the package esm_parser
                                        to the newest version of the release
                                        branch
```

And yes, `esm_versions` can upgrade itself.

11.2 Getting ESM-Versions

As was said before, if you have the Tools with a version newer than 3.1.4, you should already have `esm_versions` in your path. In case you are on an older version of the Tools, or it is missing because of problems, you need to remove the installed python packages by hand one last time, and then reinstall them using the installer:

1. Make sure to push all your local changes to branches of the repos, or save them otherwise!
2. Remove the installed python libs:

   ```
   $ rm -rf ~/.local/lib/python-whatever_your_version/site-packages/esm*
   ```

3. Remove the installed executables:

   ```
   $ rm -rf ~/.local/bin/esm*
   ```

4. Upgrade the repository esm_tools:

```
$ cd path/to/esm_tools
$ git checkout release
$ git pull
```

5. Re-install the python packages:

```
$ ./install.sh
```

You should now be on the most recent released version of the tools, and `esm_versions` should be in your `PATH`.

CHAPTER

TWELVE

ESM RUNSCRIPTS

12.1 Usage

```
esm_runscripts [-h] [-d] [-v] [-e EXPID] [-c] [-P] [-j LAST_JOBTYPE]
                   [-t TASK] [-p PID] [-x EXCLUDE] [-o ONLY]
                   [-r RESUME_FROM] [-U]
                   runscript
```

12.2 Arguments

Optional arguments	Description
-h, –help	Show this help message and exit.
-d, –debug	Print lots of debugging statements.
-v, –verbose	Be verbose.
-e EXPID, –expid EXPID	The experiment ID to use. Default `test`.
-c, –check	Run in check mode (don't submit job to supercomputer).
-P, –profile	Write profiling information (esm-tools).
-j LAST_JOBTYPE, –last_jobtype LAST_JOBTYPE	Write the jobtype this run was called from (esm-tools internal).
-t TASK, –task TASK	The task to run. Choose from: `compute`, `post`, `couple`, `tidy_and_resubmit`.
-p PID, –pid PID	The PID of the task to observe.
-x EXCLUDE, –exclude EXCLUDE	E[x]clude this step.
-o ONLY, –only ONLY	[o]nly do this step.
-r RESUME_FROM, –resume-from RESUME_FROM	[r]esume from this step.
-U, –update	[U]pdate the runscript in the experiment folder and associated files
-i, –inspect	This option can be used to [i]nspect the results of a previous run, for example one prepared with `-c`. This argument needs an additional keyword. Choose among: `overview` (gives you the same litte message you see at the beginning of each run, `lastlog` (displays the last log file), `explog` (the overall experiment logfile), `datefile` (the overall experiment logfile), `config` (the Python dict that contains all information), `size` (the size of the experiment folder), a filename or a directory name output the content of the file /directory if found in the last `run_` folder.)

12.3 Running a Model/Setup

ESM-Runscripts is the *ESM-Tools* package that allows the user to run the experiments. *ESM-Runscripts* reads the runscript (either a *bash* or *yaml* file), applies the required changes to the namelists and configuration files, submits the runs of the experiment to the compute nodes, and handles and organizes restart, output and log files. The command to run a runscript is:

```
$ esm_runscripts <runscript.yaml/.run> -e <experiment_ID>
```

The `runscript.yaml/.run` should contain all the information regarding the experiment paths, and particular configurations of the experiment (see the yaml:Runscripts section for more information about the syntax of *yaml* runscripts). The `experiment_ID` is used to identify the experiment in the scheduler and to name the experiment's directory (see *Experiment Directory Structure*). Omitting the argument `-e <experiment_ID>` will create an experiment with the default experimant ID `test`.

ESM-Runscript allows to run an experiment check by adding the `-c` flag to the previous command. This check performs all the system operations related to the experiment that would take place on a normal run (creates the experiment directory and subdirectories, copies the binaries and the necessary restart/forcing files, edits the namelists, ...) but stops before submitting the run to the compute nodes. We strongly recommend **running first a check before submitting an experiment to the compute nodes**, as the check outputs contains already valuable information to understand whether the experiment will work correctly or not (we strongly encourage users to pay particular attention to the *Namelists* and the *Missing files* sections of the check's output).

12.4 Job Phases

The following table summarizes the job phases of *ESM-Runscripts* and gives a brief description. ...

12.5 Running only part of a job

It's possible to run only part of a job. This is particularly interesting for development work; when you might only want to test a specific phase without having to run a whole simulation.

As an example; let's say you only want to run the `tidy` phase of a particular job; which will move things from the particular run folder to the overall experiment tree. In this example; the experiment will be called `test001`:

```
esm_runscripts ${PATH_TO_USER_CONFIG} -t tidy_and_resubmit
```

12.6 Experiment Directory Structure

All the files related to a given experiment are saved in the *Experiment Directory*. This includes among others model binaries, libraries, namelists, configuration files, outputs, restarts, etc. The idea behind this approach is that all the necessary files for running an experiment are contained in this folder (the user can always control through the runscript or configuration files whether the large forcing and mesh files also go into this folder), so that the experiment can be reproduced again, for example, even if there were changes into one of the model's binaries or in the original runscript.

The path of the *Experiment Directory* is composed by the `general.base_dir` path specified in the runscript (see yaml:Runscripts syntax) followed by the given `experiment_ID` during the `esm_runscripts` call:

```
<general.base_dir>/<experiment_ID>
```

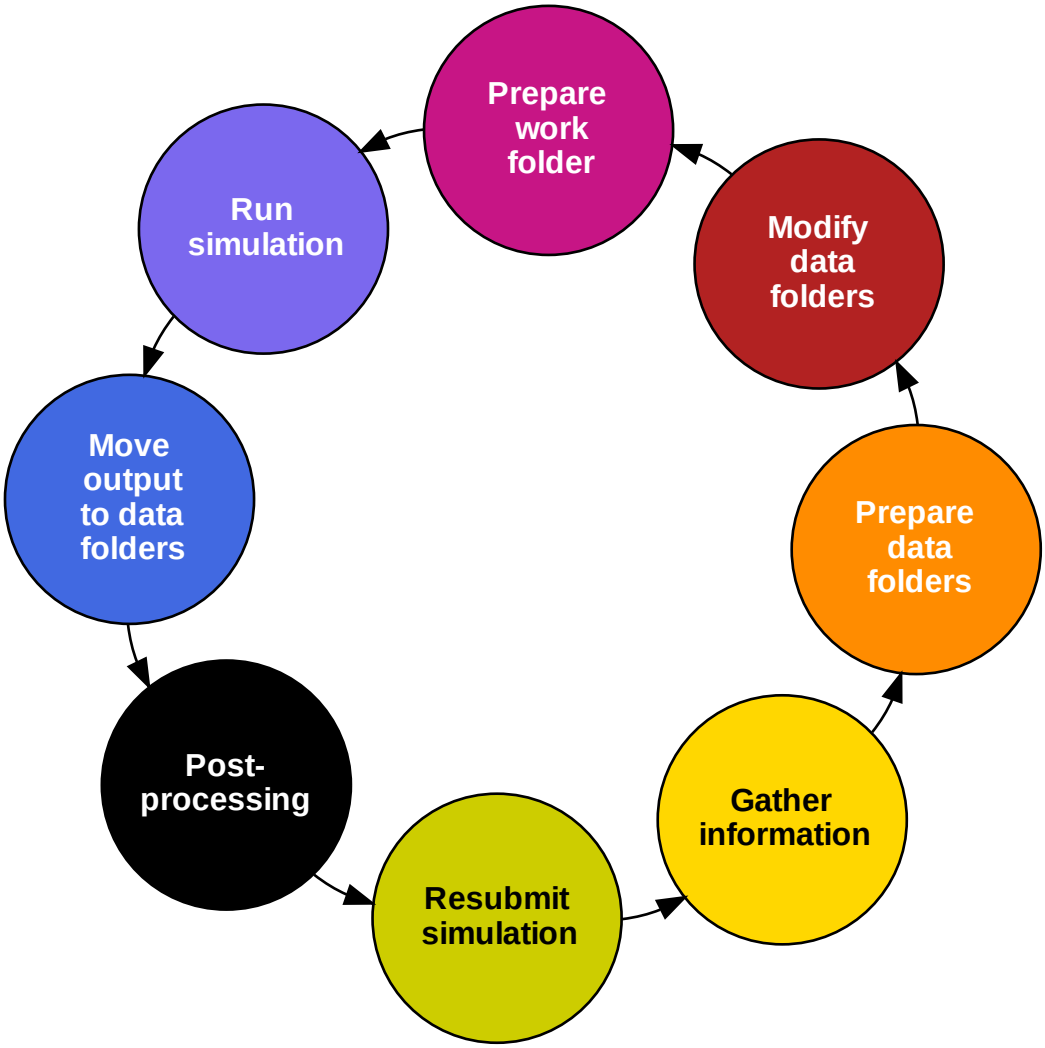

Fig. 1: ESM-Tools job phases

The **main experiment folder** (`General exp dir`) contains the subfolders indicated in the graph and table below. Each of these subfolders contains a folder for each component in the experiment (i.e. for an AWI-CM experiment the `outdata` folder will contain the subfolders `echam`, `fesom`, `hdmodel`, `jsbach`, `oasis3mct`).

The structure of the **run folder** `run_YYYYMMDD-YYYYMMDD` (`Run dir` in the graph) replicates that of the general experiment folder. *Run* directories are created before each new run and they are useful to debug and restart experiments that have crashed.

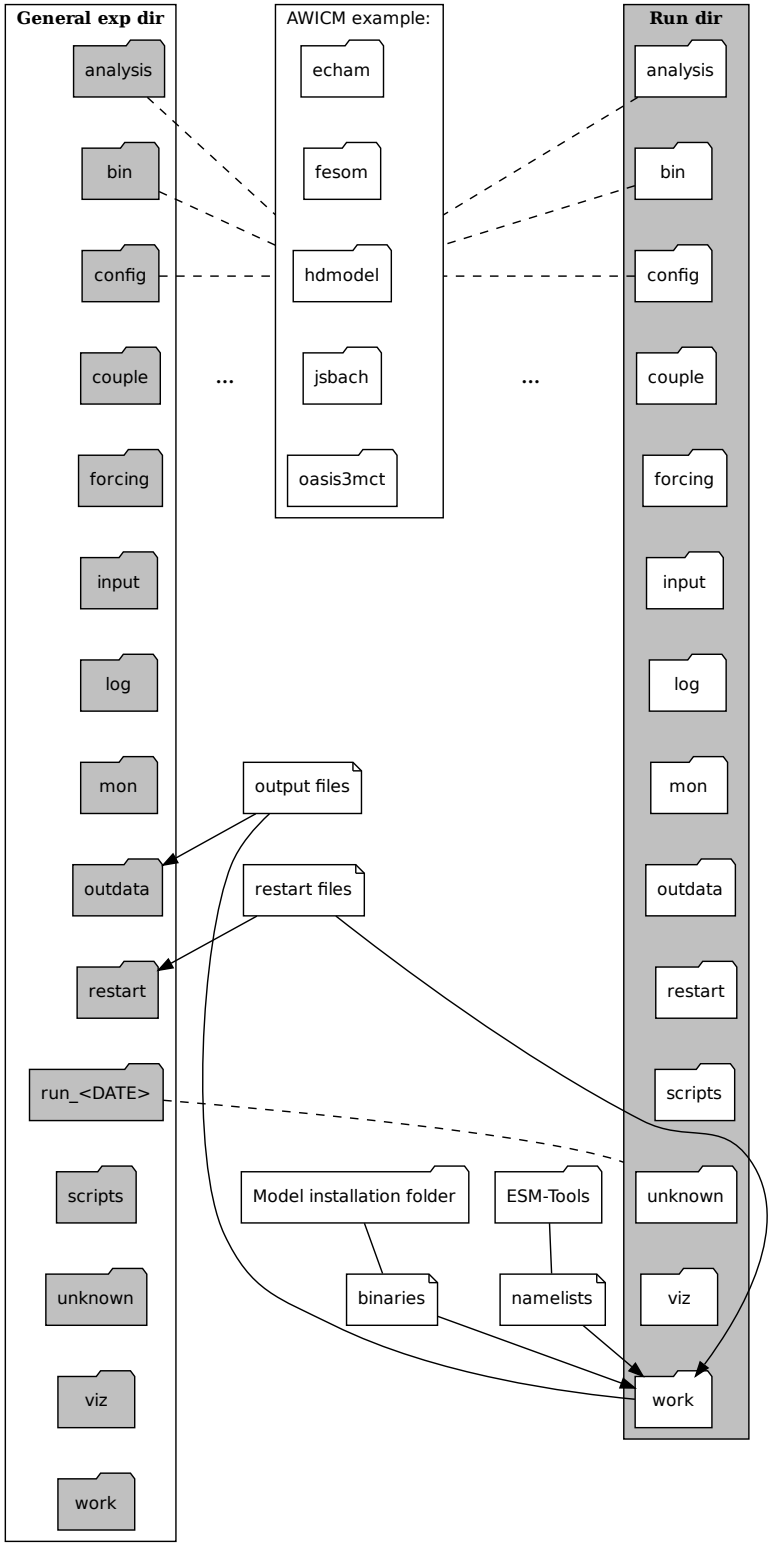

Fig. 2: Experiment directory structure

Subfolder	Files	Description
analysis	user's files	Results of user's "by-hand" analysis can be placed here.
bin	component binaries	Model binaries needed for the experiment.
config	• <experiment_ID>_finished_config.yaml • namelists • other configuration files	Configuration files for the experiment including namelists and other files specified in the component's configuration files (`<PATH>/esm_tools/configs/<component>/<component>.yaml`, see *File Dictionaries*). The file `<experiment_ID>_finished_config.yaml` is located at the base of the `config` folder and contains the whole ESM-Tools variable space for the experiment, resulting from combining the variables of the runscript, setup and component configuration files, and the machine environment file.
couple	coupling related files	Necessary files for model couplings.
forcing	forcing files	Forcing files for the experiment. Only copied here when specified by the user in the runscript or in the configuration files (*File Dictionaries*).
input	input files	Input files for the experiment. Only copied here when specified by the user in the runscript or in the configuration files (*File Dictionaries*).
log	• <experiment_ID>_<setup_name>.log • component log files	Experiment log files. The component specific log files are placed in their respective subfolder. The general log file `<experiment_ID>_<setup_name>.log` reports on the *ESM-Runscripts Job Phases* and is located at the base of the `log` folder. Log file names and copying instructions should be included in the configuration files of components (*File Dictionaries*).
mon	user's files	Monitoring scripts created by the user can be placed here.
outdata	outdata files	Outdata files are placed here. Outdata file names and copying instructions should be included in the configuration files of components (*File Dictionaries*).
restart	restart files	Restart files are placed here. Restart file names and copying instructions should be included in the configuration files of components (*File Dictionaries*).
run_YYYYMMDD-YYYYMMDD	run files	Run folder containing all the files for a given run. Folders contained here have the same names as the ones contained in the general ex-

If one file was to be copied in a directory containing a file with the same name, both files get renamed by the addition of their start date and end dates at the end of their names (i.e. `fesom.clock_YYYYMMDD-YYYYMMDD`).

Note: Having a *general* and several *run* subfolders means that files are duplicated and, when models consist of several runs, the *general* directory can end up looking very untidy. *Run* folders were created with the idea that they will be deleted once all files have been transferred to their respective folders in the *general* experiment directory. The default is not to delete this folders as they can be useful for debugging or restarting a crashed simulation, but the user can choose to delete them (see *Cleanup of run_ directories*).

12.7 Cleanup of `run_` directories

12.8 Debugging an Experiment

To debug an experiment we recommend checking the following files that you will find, either in the *general* experiment directory or in the *run* subdirectory:

- The *ESM-Tools* variable space file `config/<experiment_ID>_finished_config.yaml`.
- The run log file `run_YYYYMMDD-YYYYMMDD/<experiment_ID>_compute_YYYYMMDD-YYYYMMDD_<JobID>.log`.

For interactive debugging, you may also add the following to the `general` section of your configuration file. This will enable the pdb Python debugger, and allow you to step through the recipe.

```
general:
    debug_recipe: True
```

12.9 Setting the file movement method for filetypes in the runscript

By default, *esm_runscripts* copies all files initially into the first `run_`-folder, and from there to `work`. After the run, outputs, logs, restarts etc. are copied from `work` to `run_`, and then moved from there to the overall experiment folder. We chose that as the default setting as it is the safest option, leaving the user with everything belonging to the experiment in one folder. It is also the most disk space consuming, and it makes sense to link some files into the experiment rather than copy them.

As an example, to configure *esm_runscripts* for an echam-experiment to link the forcing and inputs, one can add the following to the runscript yaml file:

```
echam:
        file_movements:
                forcing:
                        all_directions: "link"
                input:
                        init_to_exp: "link"
                        exp_to_run: "link"
                        run_to_work: "link"
                        work_to_run: "link"
```

Both ways to set the entries are doing the same thing. It is possible, as in the `input` case, to set the file movement method independently for each of the directions; the setting `all_directions` is just a shortcut if the method is identical for all of them.

CHAPTER

THIRTEEN

ESM MOTD

The package `esm_motd` is an *ESM-Tools* integrated *message-of-the-day* system, intended as a way for the *ESM-Tools Development Team* to easily announce new releases and bug fixes to the users without the need of emailing.

It checks the versions of the different *ESM-Tools* packages installed by the user, and reports back to the user (writing to *stdout*) about packages that have available updates, and what are the new improvements that they provide (i.e. reports back that a bug in a certain package has been solved).

This check occurs every time the user uses `esm_runscripts`.

The messages, their corresponding versions and other related information is stored online in GitHub and accessed by *ESM-Tools* also online to produce the report. The user can look at this file if necessary here: *<https://github.com/esm-tools/esm_tools/tree/release/esm_tools/motd/motd.yaml>_*.

Warning: The `motd.yaml` file is to be modified exclusively by the ESM-Tools Core Development Team, so... stay away from it ;-)

CHAPTER

FOURTEEN

COOKBOOK

In this chapter you can find multiple recipes for different ESM-Tools functionalities, such running a model, adding forcing files, editing defaults in namelists, etc.

If you'd like to contribute with your own recipe, or ask for a recipe, please open a documentation issue on our GitHub repository.

Note: Throughout the cookbook, we will sometimes refer to a nested part of a configuration via dot notation, e.g. `a.b.c`. Here, we mean the following in a YAML config file:

```yaml
a:
  b:
    c: "foo"
```

This would indicate that the value of `a.b.c` is `"foo"`. In Python, you would access this value as `a["b"]["c"]`.

14.1 Change/Add Flags to the sbatch Call

Feature available since version: 4.2

If you are using *SLURM* batch system together with *ESM-Tools* (so far the default system), you can modify the `sbatch` call flags by modifying the following variables from your runscript, inside the `computer` section:

Key	Description
mail_type, mail_user	Define these two variables to get updates about your slurm-job through email.
single_proc_submit_flag	By default defined as `--ntasks-per-node=1`
additional_flags	To add any additional flag that is not predefined in *ESM-Tools*

14.1.1 Example

Assume you want to run a simulation using the Quality of Service flag (`--qos`) of *SLURM* with value `24h`. Then, you'll need to define the `additional_flags` inside the `computer` section of your runscript. This can be done by adding the following to your runscript:

```yaml
computer:
    additional_flags: "--qos=24h"
```

14.2 Applying a temporary disturbance to ECHAM to overcome numeric instability (lookup table overflows of various kinds)

Feature available since version: `esm_runscripts v4.2.1`

From time to time, the ECHAM family of models runs into an error resulting from too high wind speeds. This may look like this in your log files:

```
30: ================================================================================
30:
30: FATAL ERROR in cuadjtq (1):  lookup table overflow
30:  FINISH called from PE: 30
```

To overcome this problem, you can apply a small change to the factor "by which stratospheric horizontal diffussion is increased from one level to the next level above." (`mo_hdiff.f90`), that is the namelist parameter `enstdif`, in the `dynctl` section of the ECHAM namelist. As this is a common problem, there is a way to have the run do this for specific years of your simulation. Whenever a model year crashes due to numeric instability, you have to apply the method outlined below.

1. Generate a file to list years you want disturbed.

   In your experiment script folder (**not** the one specific for each run), you can create a file called `disturb_years.dat`. An abbreviated file tree would look like:

2. Add years you want disturbed.

   The file should contain a list of years the disturbance should be applied to, seperated by new lines. In practice, you will add a new line with the value of the model year during which the model crashes whenever such a crash occurs.

14.2.1 Example

In this example, we disturb the years 2005, 2007, and 2008 of an experiment called `EXAMPLE` running on `ollie`:

```
$ cat /work/ollie/pgierz/test_esmtools/EXAMPLE/scripts/disturb_years.dat
2005
2007
2008
```

You can also set the disturbance strength in your configuration under `echam.disturbance`. The default is `1.000001`. Here, we apply a 200% disturbance whenever a "disturb_year" is encountered.

```
echam:
    disturbance: 2.0
```

14.2.2 See also

- ECHAM6 User Handbook, Table 2.4, dynctl
- Relevant source code

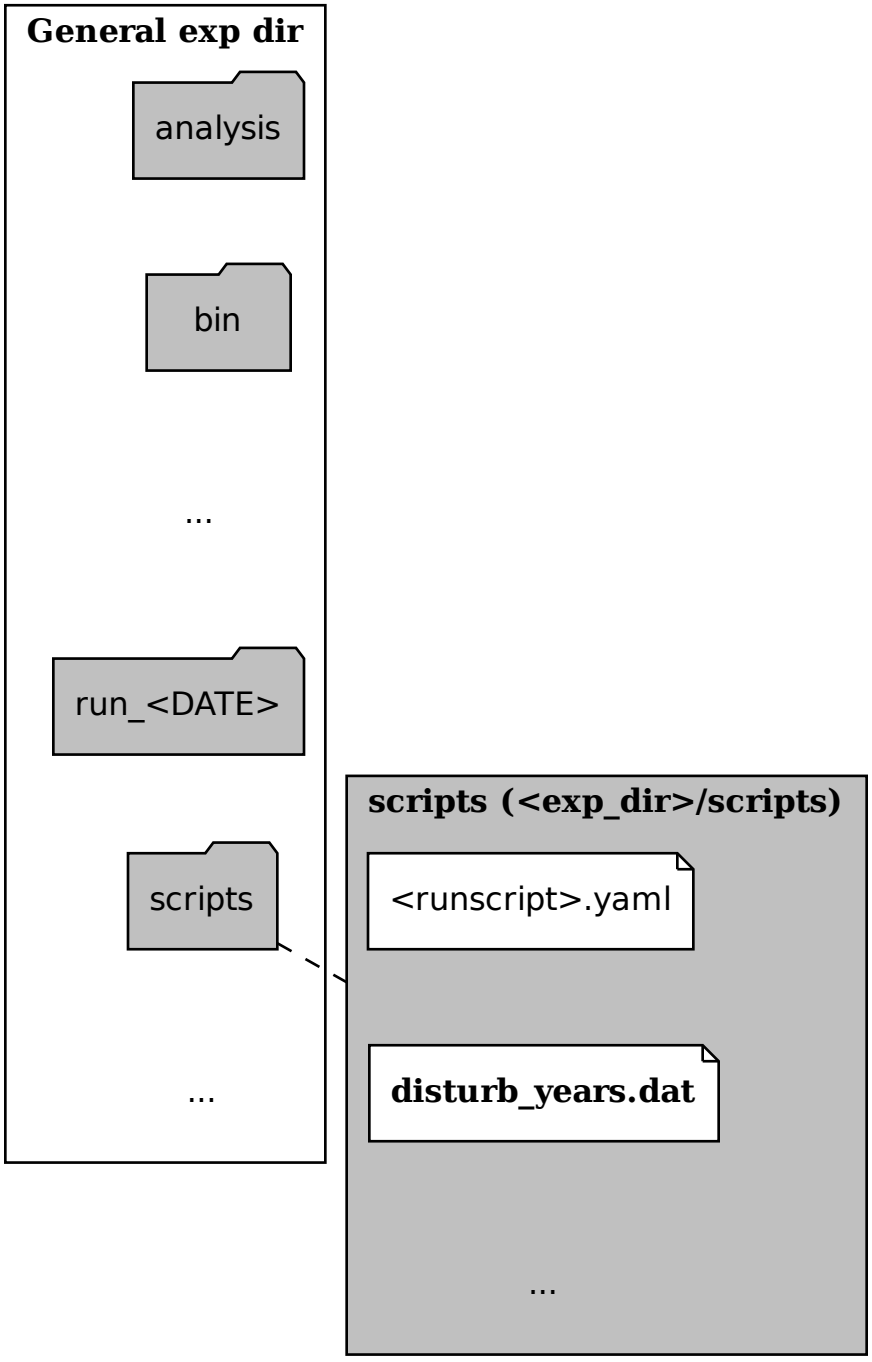

Fig. 1: disturb_years.dat location

14.3 Changing Namelist Entries from the Runscript

Feature available since version: 4.2

You can modify namelists directly from your user yaml runscript configuration.

1. Identify which namelist you want to modify and ensure that it is in the correct section. For example, you can only modify `ECHAM` specific namelists from an `ECHAM` block.
2. Find the subsection ("chapter") of the namelist you want to edit.
3. Find the setting ("key") you want to edit
4. Add a `namelist_changes` block to your configuration, specify next the namelist filename you want to modify, then the chapter, then the key, and finally the desired value.

In dot notation, this will look like: `<model_name>.namelist_changes.<namelist_name>.<chapter_name>.<key_name> = <value>`

14.3.1 Example

Here are examples for just the relevant YAML change, and for a full runscript using this feature.

Snippet

Full Runscript

In this example, we modify the `co2vmr` of the `radctl` section of `namelist.echam`.

```
echam:
    namelist_changes:
        namelist.echam:
            radctl:
                co2vmr: 1200e-6
```

In this example, we set up AWI-ESM 2.1 for a 4xCO2 simulation. You can see how multiple namelist changes are applied in one block.

```
general:
    setup_name: "awiesm"
    compute_time: "02:30:00"
    initial_date: "2000-01-01"
    final_date: "2002-12-31"
    base_dir: "/work/ab0246/a270077/For_Christian/experiments/"
    nmonth: 0
    nyear: 1
    account: "ab0246"

echam:
    restart_unit: "years"
    nprocar: 0
    nprocbr: 0
    namelist_changes:
            namelist.echam:
                    radctl:
                            co2vmr: 1137.e-6
                    parctl:
                            nprocar: 0
                            nprocbr: 0
```

(continues on next page)

(continued from previous page)

```
                    runctl:
                            default_output: True

awiesm:
    version: "2.1"
    postprocessing: true
    scenario: "PALEO"
    model_dir: "/work/ab0246/a270077/For_Christian/model_codes/awiesm-2.1/"

fesom:
    version: "2.0"
    res: "CORE2"
    pool_dir: "/pool/data/AWICM/FESOM2"
    mesh_dir: "/work/ba1066/a270061/mesh_CORE2_finaltopo_mean/"
    restart_rate: 1
    restart_unit: "y"
    restart_first: 1
    lresume: 0
    namelist_changes:
        namelist.config:
            paths:
                ClimateDataPath: "/work/ba0989/a270077/AWIESM_2_1_LR_concurrent_rad/
↪nonstandard_input_files/fesom/hydrography/"

jsbach:
    input_sources:
        jsbach_1850: "/work/ba1066/a270061/mesh_CORE2_finaltopo_mean/tarfilesT63/
↪input/jsbach/jsbach_T63CORE2_11tiles_5layers_1850.nc"
```

14.3.2 Practical Usage

It is generally a good idea to run your simulation once in **check** mode before actually submitting and examining the resulting namelists:

```
$ esm_runscripts <your_config.yaml> -e <expid> -c
```

The namelists are printed in their final form as part of the log during the job submission and can be seen on disk in the `work` folder of your first `run_XZY` folder.

Note that you can have several chapters for one namelist or several namelists included in one `namelist_changes` block, but you can only have one `namelist_changes` block per model or component (see *Changing Namelists*).

14.3.3 See also

- Default namelists on GitHub
- *Changing Namelists*
- *What Is YAML?*

14.4 How to setup runscripts for different kind of experiments

This recipe describes how to setup a runscript for the following different kinds of experiments. Besides the variables described in *ESM-Tools Variables*, add the following variables to your runscript, as described below.

- **Initial run**: An experiment from initial model conditions.

```
general:
    lresume: 0
```

- **Restart**: An experiment that restarts from a previous experiment with the same experiment id.

```
general:
    lresume: 1
```

- **Branching off**: An experiment that restarts from a previous experiment but with a different experiment id.

```
general:
    lresume: 1
    ini_parent_exp_id: <old-experiment-id>
    ini_restart_dir: <path-to-restart-dir-of-old-experiment>/restart/
```

- **Branching off and redate**: An experiment that restarts from a previous experiment with a different experiment id and if this experiment should be continued with a diiferent start date.

```
general:
    lresume: 1
    ini_parent_exp_id: <old-experiment-id>
    ini_restart_dir: <path-to-restart-dir-of-old-experiment>/restart/
    first_initial_year: <year>
```

14.4.1 See also

- *ESM-Tools Variables*
- *What Is YAML?*

14.5 Implement a New Model

Feature available since version: 4.2

1. Upload your model into a repository such us *gitlab.awi.de*, *gitlab.dkrz.de* or *GitHub*. Make sure to set up the right access permissions, so that you comply with the licensing of the software you are uploading.
2. If you are interested in implementing more than one version of the model, we recommend you to commit them to the master branch in the order they were developed, and that you create a tag per version. For example:
   a. Clone the empty master branch you just created and add your model files to it:

```
$ git clone https://<your_repository>
$ cp -rf <your_model_files_for_given_version> <your_repository_folder>
$ git add .
```

   b. Commit, tag the version and push the changes to your repository:

```
$ git commit -m "your comment here"
$ git tag -a <version_id> -m "your comment about the version"
$ git push -u origin <your_master_branch>
$ git push origin <version_id>
```

c. Repeat steps *a* and *b* for all the versions that you would like to be present in ESM-Tools.

3. Now that you have your model in a repository you are ready to implement it into *esm_tools*. First, you will need to create your own branch of *esm_tools*, following the steps 1-4 in *Contribution to esm_tools Package*. The recommended name for the branch would be `feature/<name_of_your_model>`.

4. Then you will need to create a folder for your model inside `esm_tools/configs/components` and create the model's *yaml* file:

```
$ mkdir <PATH>/esm_tools/configs/components/<model>
$ touch <PATH>/esm_tools/configs/components/<model>/<model>.yaml
```

5. Use your favourite text editor to open and edit your `<model>.yaml` in the `esm_tools/configs/components/<model>` folder:

```
$ <your_text_editor> <PATH>/esm_tools/configs/components/<model>/<model>.yaml
```

6. Complete the following information about your model:

```
# YOUR_MODEL YAML CONFIGURATION FILE
#

model: your_model_name
type: type_of_your_model       # atmosphere, ocean, etc.
version: "the_default_version_of_your_model"
```

7. Include the names of the different versions in the `available_versions` section and the compiling information for the default version:

```
[...]

available_versions:
- "1.0.0"
- "1.0.1"
- "1.0.2"
git-repository: "https://your_repository.git"
branch: your_model_branch_in_your_repo
install_bins: "path_to_the_binaries_after_comp"
comp_command: "your_shell_commands_for_compiling"      # You can use the defaults "
↪${defaults.comp_command}"
clean_command: "your_shell_commands_for_cleaning"      # You can use the defaults "
↪${defaults.clean_command}"

executable: your_model_command

setup_dir: "${model_dir}"
bin_dir: "${setup_dir}/name_of_the_binary"
```

In the `install_bins` key you need to indicate the path inside your model folder where the binaries are compiled to, so that *esm_master* can find them once compiled. The `available_versions` key is needed for *esm_master* to list the versions of your model. The `comp_command` key indicates the command needed to compile your model, and can be set as `${defaults.comp_command}` for a default command (`mkdir -p`

build; cd build; cmake ..; make install -j `nproc --all`), or you can define your own list of compiling commands separated with ; ("command1; command2").

8. At this point you can choose between including all the version information inside the same <model>.yaml file, or to distribute this information among different version files:

   Single file

   Multiple version files

   In the <model>.yaml, use a choose_ switch (see *Switches (choose_)*) to modify the default information that you added in step 7 to meet the requirements for each specific version. For example, each different version has its own git branch:

```
choose_version:
        "1.0.0":
                branch: "1.0.0"
        "1.0.1":
                branch: "1.0.1"
        "1.0.2":
                branch: "develop"
```

   a. Create a *yaml* file per version or group of versions. The name of these files should be the same as the ones in the available_versions section, in the main <model>.yaml file or, in the case of a file containing a group of versions, the shared name among the versions (i.e. fesom-2.0.yaml):

```
$ touch <PATH>/esm_tools/configs/<model>/<model-version>.yaml
```

   b. Open the version file with your favourite editor and include the version specific changes. For example, you want that the version 1.0.2 from your model pulls from the develop git branch, instead of from the default branch. Then you add to the <model>-1.0.2.yaml version file:

```
branch: "develop"
```

   Another example is the fesom-2.0.yaml. While fesom.yaml needs to contain all available_versions, the version specific changes are split among fesom.yaml (including information about versions 1) and fesom-2.0.yaml (including information about versions 2):

   fesom.yaml

   fesom-2.0.yaml

```
[ ... ]

available_versions:
- 2.0-o
- 2.0-esm-interface
- '1.4'
- '1.4-recom-mocsy-slp'
- 2.0-esm-interface-yac
- 2.0-paleodyn
- 1.4-recom-awicm
- '2.0'
- '2.0-r' # OG: temporarily here
choose_version:
  '1.4-recom-awicm':
    destination: fesom-1.4
    branch: co2_coupling
  '1.4-recom-mocsy-slp':
```

(continues on next page)

(continued from previous page)

```
    branch: fesom-recom-mocsy-slp
    destination: fesom-1.4

[ ... ]
```

```
[ ... ]

choose_version:
  '2.0':
    branch: 2.0.2
    git-repository:
    - https://gitlab.dkrz.de/FESOM/fesom2.git
    - github.com/FESOM/fesom2.git
    install_bins: bin/fesom.x
  2.0-esm-interface:
    branch: fesom2_using_esm-interface
    destination: fesom-2.0
    git-repository:
    - https://gitlab.dkrz.de/a270089/fesom-2.0_yac.git
    install_bins: bin/fesom.x

[ ... ]
```

Note: These are just examples of model configurations, but the parser used by *ESM-Tools* to read *yaml* files (*esm_parser*) allows for a lot of flexibility in their configuration; i.e., imagine that the different versions of your model are in different repositories, instead of in different branches, and their paths to the binaries are also different. Then you can include the `git-repository` and `install_bins` variables inside the corresponding version case for the `choose_version`.

9. You can now check if *esm_master* can list and install your model correctly:

```
$ esm_master
```

This command should return, without errors, a list of available models and versions including yours. Then you can actually try installing your model in the desired folder:

```
$ mkdir ~/model_codes
$ cd ~/model_codes
$ esm_master install-your_model-version
```

10. If everything works correctly you can check that your changes pass `flake8`:

```
$ flake8 <PATH>/esm_tools/configs/components/<model>/<model>.yaml
```

Use this link to learn more about `flake8` and how to install it.

11. Commit your changes, push them to the `origin` remote repository and submit a pull request through GitHub (see steps 5-7 in *Contribution to esm_tools Package*).

Note: You can include all the compiling information inside a `compile_infos` section to avoid conflicts with other `choose_version` switches present in your configuration file.

14.5.1 See also

- *ESM-Tools Variables*
- *Switches (choose_)*
- *What Is YAML?*

14.6 Implement a New Coupled Setup

Feature available since version: 4.2

An example of the different files needed for *AWICM* setup is included at the end of this section (see recipes/add_model_setup:Example).

1. Make sure the models, couplers and versions you want to use, are already available for *esm_master* to install them (`$ esm_master` and check the list). If something is missing you will need to add it following the instructions in *Implement a New Model*.
2. Once everything you need is available to *esm_master*, you will need to create your own branch of *esm_tools*, following the steps 1-4 in *Contribution to esm_tools Package*.
3. Setups need two types of files: 1) **coupling files** containing information about model versions and coupling changes, and 2) **setup files** containing the general information about the setup and the model changes. In this step we focus on the creation of the **coupling files**.
   a. Create a folder for your couplings in `esm_tools/configs/couplings`:

   ```
   $ cd esm_tools/configs/couplings/
   $ mkdir <coupling_name1>
   $ mkdir <coupling_name2>
   ...
   ```

   The naming convention we follow for the coupling files is `component1-version+component2-version+...`

   b. Create a *yaml* file inside the coupling folder with the same name:

   ```
   $ touch <coupling_name1>/<coupling_name1>.yaml
   ```

   c. Include the following information in each coupling file:

   ```
   components:
   - "model1-version"
   - "model2-version"
   - [ ... ]
   - "coupler-version"
   coupling_changes:
   - sed -i '/MODEL1_PARAMETER/s/OFF/ON/g' model1-1.0/file_to_change
   - sed -i '/MODEL2_PARAMETER/s/OFF/ON/g' model2-1.0/file_to_change
   - [ ... ]
   ```

   The `components` section should list the models and couplers used for the given coupling including, their required version. The `coupling_changes` subsection should include a list of commands to make the necessary changes in the component's compilation configuration files (`CMakeLists.txt`, `configure`, etc.), for a correct compilation of the coupled setup.
4. Now, it is the turn for the creation of the **setup file**. Create a folder for your coupled setup inside `esm_tools/configs/setups` folder, and create a *yaml* file for your setup:

```
$ mkdir <PATH>/esm_tools/configs/setups/<your_setup>
$ touch <PATH>/esm_tools/configs/setups/<your_setup>/<setup>.yaml
```

5. Use your favourite text editor to open and edit your `<setup>.yaml` in the `esm_tools/configs/setups/<your_setup>` folder:

```
$ <your_text_editor> <PATH>/esm_tools/configs/setups/<your_setup>/<setup>.yaml
```

6. Complete the following information about your setup:

```
##############################################################################
↪#######
######################### NAME_VERSION YAML CONFIGURATION FILE ################
↪#######
##############################################################################
↪#######

general:
        model: your_setup
        version: "your_setup_version"

        coupled_setup: True

        include_models:            # List of models, couplers and componentes of␣
↪the setup.
                - component_1      # Do not include the version number
                - component_2
                - [ ... ]
```

Note: *Models* do not have a `general` section but in the *setups* the `general` section is mandatory.

7. Include the names of the different versions in the `available_versions` section:

```
general:

        [ ... ]

        available_versions:
                - "1.0.0"
                - "1.0.1"
```

The `available_versions` key is needed for *esm_master* to list the versions of your setup.

8. In the `<setup>.yaml`, use a `choose_` switch (see *Switches (choose_)*) to assign the coupling files (created in step 3) to their corresponding setup versions:

```
general:

    [ ... ]

    choose_version:
            "1.0.0":
                    couplings:
                            - "model1-1.0+model2-1.0"
            "1.0.1":
                    couplings:
```

(continues on next page)

(continued from previous page)

```
                                - "model1-1.1+model2-1.1"

    [ ... ]
```

9. You can now check if *esm_master* can list and install your coupled setup correctly:

```
$ esm_master
```

This command should return, without errors, a list of available setups and versions including yours. Then you can actually try installing your setup in the desire folder:

```
$ mkdir ~/model_codes
$ cd ~/model_codes
$ esm_master install-your_setup-version
```

10. If everything works correctly you can check that your changes pass `flake8`:

```
$ flake8 <PATH>/esm_tools/configs/setups/<your_setup>/<setup>.yaml
$ flake8 <PATH>/esm_tools/configs/couplings/<coupling_name>/<coupling_name>.yaml
```

Use this link to learn more about `flake8` and how to install it.

11. Commit your changes, push them to the `origin` remote repository and submit a pull request through GitHub (see steps 5-7 in *Contribution to esm_tools Package*).

14.6.1 Example

Here you can have a look at relevant snippets of some of the *AWICM-1.0* files.

fesom-1.4+echam-6.3.04p1.yaml

awicm.yaml

One of the coupling files for *AWICM-1.0* (`esm_tools/configs/couplings/fesom-1.4+echam-6.3.04p1/fesom-1.4+echam-6.3.04p1.yaml`):

```
components:
- echam-6.3.04p1
- fesom-1.4
- oasis3mct-2.8
coupling_changes:
- sed -i '/FESOM_COUPLED/s/OFF/ON/g' fesom-1.4/CMakeLists.txt
- sed -i '/ECHAM6_COUPLED/s/OFF/ON/g' echam-6.3.04p1/CMakeLists.txt
```

Setup file for *AWICM* (`esm_tools/configs/setups/awicm/awicm.yaml`):

```
#############################################################################
↪###
######################### AWICM 1 YAML CONFIGURATION FILE ###################
↪###
#############################################################################
↪###



general:
```

(continues on next page)

(continued from previous page)

```
        model: awicm
        #model_dir: ${esm_master_dir}/awicm-${version}

        coupled_setup: True

        include_models:
                - echam
                - fesom
                - oasis3mct

        version: "1.1"
        scenario: "PI-CTRL"
        resolution: ${echam.resolution}_${fesom.resolution}
        postprocessing: false
        post_time: "00:05:00"
        choose_general.resolution:
                T63_CORE2:
                        compute_time: "02:00:00"
                T63_REF87K:
                        compute_time: "02:00:00"
                T63_REF:
                        compute_time: "02:00:00"
        available_versions:
        - '1.0'
        - '1.0-recom'
        - CMIP6
        choose_version:
          '1.0':
            couplings:
            - fesom-1.4+echam-6.3.04p1
          '1.0-recom':
            couplings:
            - fesom-1.4+recom-2.0+echam-6.3.04p1
          CMIP6:
            couplings:
            - fesom-1.4+echam-6.3.04p1
```

14.6.2 See also

- *ESM-Tools Variables*
- *Switches (choose_)*
- *What Is YAML?*

14.7 Include a New Forcing/Input File

Feature available since version: 4.2

There are several ways of including a new forcing or input file into your experiment depending on the degree of control you'd like to achieve. An important clarification is that `<forcing/input>_sources` file dictionary specifies the **sources** (paths to the files in the pools or personal folders, that need to be copied or linked into the experiment folder). On the other hand `<forcing/input>_files` specifies which of these sources are to be **included in the experiment**. This allows us to have many sources already available to the user, and then the user can simply choose which of them to use by chosing from `<forcing/input>_files`. `<forcing/input>_in_work` is used to copy the files into the work folder (`<base_dir>/<exp_id>/run_<DATE>/work`) if necessary and change their name. For more technical details see *File Dictionaries*.

The next sections illustrate some of the many options to handle forcing and input files.

14.7.1 Source Path Already Defined in a Config File

1. Make sure the source of the file is already specified inside the `forcing_sources` or `input_sources` *file dictionaries* in the configuration file of the setup or model you are running, or on the `further_reading` files.
2. In your runscript, include the *key* of the source file you want to include inside the `forcing_files` or `input_files` section.

Note: Note that the *key* containing the source in the `forcing_sources` or `input_sources` can be different than the key specified in `forcing_files` or `input_files`.

Example

ECHAM

In ECHAM, the source and input file paths are specified in a separate file (`<PATH>/esm_tools/configs/components/echam/echam.datasets.yaml`) that is reached through the `further_reading` section of the `echam.yaml`. This file includes a large number of different sources for input and forcing contained in the pool directories of the HPC systems Ollie and Mistral. Let's have a look at the `sst` forcing file options available in this file:

```
forcing_sources:
        # sst
        "amipsst":
                "${forcing_dir}/amip/${resolution}_amipsst_@YEAR@.nc":
                        from: 1870
                        to: 2016
        "pisst": "${forcing_dir}/${resolution}${ocean_resolution}_piControl-LR_sst_
↪1880-2379.ncy"
```

This means that from our runscript we will be able to select either `amipsst` or `pisst` as *sst* forcing files. If you define `scenario` in *ECHAM* be `PI-CTRL` the correct file source (`pisst`) is already selected for you. However, if you would like to select this file manually you can just simply add the following to your runscript:

```
forcing_files:
        sst: pisst
```

14.7.2 Modify the Source of a File

To change the path of the source for a given forcing or input file from your runscript:

1. Include the source path under a *key* inside `forcing_sources` or `input_sources` in your runscript:

```
<forcing/input>_sources:
        <key_for_your_file>: <path_to_your_file>
```

   If the source is not a single file, but there is a file per year use the `@YEAR@` and `from: to:` functionality in the path to copy only the files corresponding to that run's year:

```
<forcing/input>_sources:
        <key_for_your_source>: <firt_part_of_the_path>@YEAR@<second_part_of_the_
↪path>
                from: <first_year>
                to: <last_year>
```

2. Make sure the *key* for your path is defined in one of the config files that you are using, inside of either `forcing_files` or `input_files`. If it is not defined anywhere you will have to include it in your runscript:

```
<forcing/input>_files:
        <key_for_your_file>: <key_for_your_source>
```

14.7.3 Copy the file in the work folder and/or rename it

To copy the files from the forcing/input folders into the work folder (`<base_dir>/<exp_id>/run_<DATE>/work`) or rename them:

1. Make sure your file and its source is defined somewhere (either in the config files or in your runscript) in `<forcing/input>_sources` and `<forcing/input>_files` (see subsections *Source Path Already Defined in a Config File* and *Modify the Source of a File*).
2. In your runscript, add the *key* to the file you want to **copy** with *value* the same as the *key*, inside <forcing/input>_in_work:

```
<forcing/input>_in_work:
        <key_for_your_file>: <key_for_your_file>
```

3. If you want to **rename** the file set the *value* to the desired name:

```
<forcing/input>_in_work:
        <key_for_your_file>: <key_for_your_file>
```

Example

ECHAM

In *ECHAM* the `sst` forcing file depends in the scenario defined by the user:

esm_tools/config/component/echam/echam.datasets.yaml

```
forcing_sources:
        # sst
        "amipsst":
                "${forcing_dir}/amip/${resolution}_amipsst_@YEAR@.nc":
                        from: 1870
                        to: 2016
        "pisst": "${forcing_dir}/${resolution}${ocean_resolution}_piControl-LR_sst_
↪1880-2379.nc"
```

esm_tools/config/component/echam/echam.yaml

```
choose_scenario:
        "PI-CTRL":
                forcing_files:
                        sst: pisst
                        [ ... ]
```

If `scenario: "PI-CTRL"` then the source selected will be `${forcing_dir}/${resolution}${ocean_resolution}_piControl-LR_sst_1880-2379.nc` and the name of the file copied to the experiment forcing folder will be `${resolution}${ocean_resolution}_piControl-LR_sst_1880-2379.nc`. However, *ECHAM* needs this file in the same folder as the binary (the `work` folder) under the name `unit.20`. To copy and rename this file into the `work` folder the following lines are used in the `echam.yaml` configuration file:

```
forcing_in_work:
        sst: "unit.20"
```

You can use the same syntax **inside your runscript** to copy into the `work` folder any forcing or input file, and rename it.

14.7.4 See also

- *What Is YAML?*
- *File Dictionaries*

14.8 Exclude a Forcing/Input File

Feature available since version: 4.2

To exclude one of the predefined forcing or input files from being copied to your experiment folder:

1. Find the *key* of the file to be excluded inside the config file, `<forcing/input>_files` *file dictionary*.
2. In your runscript, use the `remove_` functionality to exclude this *key* from the `<forcing/input>_files` *file dictionary*:

```
remove_<input/forcing>_files:
        - <key_of_the_file1>
        - <key_of_the_file2>
        - ...
```

14.8.1 Example

ECHAM

To exclude the `sst` forcing file from been copied to the experiment folder include the following lines in your runscript:

```
remove_forcing_files:
        - sst
```

14.8.2 See also

- *What Is YAML?*
- *Remove Elements from a List/Dictionary (remove_)*
- *File Dictionaries*

14.9 Using your own namelist

Feature available since version: 4.2

> **Warning:** This feature is only recommended if the number of changes that need to be applied to the default namelist is very large, otherwise we recommend to use the feature `namelist_changes` (see *Changing Namelist Entries from the Runscript*). You can check the default namelists here.

In your runscript, you can instruct *ESM-Tools* to substitute a given default namelist by a namelist of your choice.

1. Search for the `config_sources` variable inside the configuration file of the model you are trying to run, and then, identify the "key" containing the path to the default namelist.
2. In your runscript, indented in the corresponding model section, add an `add_config_sources` section, containing a variable whose "key" is the one of step 1, and the value is the path of the new namelist.
3. Bare in mind, that namelists are first loaded by *ESM-Tools*, and then modified by the default `namelist_changes` in the configuration files. If you want to ignore all those changes for the your new namelist you'll need to add `remove_namelist_changes: [<name_of_your_namelist>]`.

In dot notation both steps will look like: `<model_name>.<add_config_sources>.<key_of_the_namelist>: <path_of_your_namelist>` `<model_name>.<remove_namelis_changes>: [<name_of_your_namelist>]`

> **Warning:** Use step 3 at your own risk! Many of the model specific information and functionality is transferred to the model through `namelist_changes`, and therefore, we discourage you from using `remove_namelist_changes` unless you have a very deep understanding of the configuration file and the model. Following *Changing Namelist Entries from the Runscript* would be a safest solution.

14.9.1 Example

In this example we show how to use an *ECHAM* `namelist.echam` and a *FESOM* `namelist.ice` that are not the default ones and omit the `namelist_changes` present in `echam.yaml` and `fesom.yaml` configuration files.

ECHAM

FESOM

Following step 1, search for the `config_sources` dictionary inside the `echam.yaml`:

```
# Configuration Files:
config_sources:
        "namelist.echam": "${namelist_dir}/namelist.echam"
```

In this case the "key" is `"namelist.echam"` and the "value" is `"${namelist_dir}/namelist.echam"`. Let's assume your namelist is in the directory `/home/ollie/<usr>/my_namelists`. Following step 2, you will need to include the following in your runscript:

```
echam:
        add_config_sources:
                "namelist.echam": /home/ollie/<usr>/my_namelists/namelist.echam
```

If you want to omit the `namelist_changes` in `echam.yaml` or any other configuration file that your model/couple setup is using, you'll need to add to your runscript `remove_namelist_changes: [namelist.echam]` (step 3):

```
echam:
        add_config_sources:
                "namelist.echam": /home/ollie/<usr>/my_namelists/namelist.echam

        remove_namelist_changes: [namelist.echam]
```

Warning: Many of the model specific information and functionality is transferred to the model through `namelist_changes`, and therefore, we discourage you from using this unless you have a very deep understanding of the `echam.yaml` file and the ECHAM model. For example, using `remove_namelist_changes: [namelist.echam]` will destroy the following lines in the `echam.yaml`:

```
choose_lresume:
        False:
                restart_in_modifications:
                        "[[streams-->STREAM]]":
                            - "vdate <--set_global_attr-- ${start_date!syear!smonth!
↪sday}"
                              # - fdate "<--set_dim--" ${year_before_date}
                              # - ndate "<--set_dim--" ${steps_in_year_before}
        True:
                # pseudo_start_date: $(( ${start_date} - ${time_step} ))
                add_namelist_changes:
                        namelist.echam:
                                runctl:
                                        dt_start: "remove_from_namelist"
```

This lines are relevant for correctly performing restarts, so if `remove_namelist_changes` is used, make sure to have the approrpiate commands on your runscript to remove `dt_start` from your namelist in case of a restart.

Following step 1, search for the `config_sources` dictionary inside the `fesom.yaml`:

```
config_sources:
        config:  "${namelist_dir}/namelist.config"
        forcing: "${namelist_dir}/namelist.forcing"
        ice:     "${namelist_dir}/namelist.ice"
        oce:     "${namelist_dir}/namelist.oce"
        diag:    "${namelist_dir}/namelist.diag"
```

In this case the "key" is `ice` and the "value" is `${namelist_dir}/namelist.ice`. Let's assume your namelist is in the directory `/home/ollie/<usr>/my_namelists`. Following step 2, you will need to include the following in your runscript:

```
fesom:
        add_config_sources:
                ice: "/home/ollie/<usr>/my_namelists/namelist.ice"
```

If you want to omit the `namelist_changes` in `fesom.yaml` or any other configuration file that your model/couple setup is using, you'll need to add to your runscript `remove_namelist_changes: [namelist.ice]` (step 3):

```
fesom:
        add_config_sources:
                ice: "/home/ollie/<usr>/my_namelists/namelist.ice"

        remove_namelist_changes: [namelist.ice]
```

Warning: Many of the model specific information and functionality is transferred to the model through `namelist_changes`, and therefore, we discourage you from using this unless you have a very deep understanding of the `fesom.yaml` file and the FESOM model.

14.9.2 See also

- Default namelists on GitHub
- *Append to an Existing List (add_)*
- *Changing Namelists*
- *What Is YAML?*

14.10 How to branch-off FESOM from old spinup restart files

When you branch-off from very old FESOM ocean restart files, you may encounter the following runtime error:

```
read ocean restart file
Error:
NetCDF: Invalid dimension ID or name
```

This is because the naming of the NetCDF time dimension variable in the restart file has changed from `T` to `time` during the development of *FESOM* and the different *FESOM* versions. Therefore, recent versions of *FESOM* expect the name of the time dimension to be `time`.

In order to branch-off experiments from spinup restart files that use the old name for the time dimension, you need to rename this dimension before starting the branch-off experiment.

Warning: The following work around will change the restart file permanently. Make sure you do not apply this to the original file.

To rename a dimension variable of a NetCDF file, you can use `ncrename`:

```
ncrename -d T,time <copy_of_restart_spinup_file>.nc
```

where `T` is the old dimension and `time` is the new dimension.

14.10.1 See also

- cookbook:How to run a branch-off experiment

CHAPTER

FIFTEEN

FREQUENTLY ASKED QUESTIONS

15.1 Installation

1. **Q**: My organization is not in the pull-down list I get when trying the Federated Login to gitlab.awi.de.

   A: Then maybe your institution just didn't join the DFN-AAI. You can check that at https://tools.aai.dfn.de/entities/.

2. **Q**: I am trying to use the Federated Login, and that seems to work fine. When I should be redirected to the gitlab server though, I get the error that my uid is missing.

   A: Even though your organization joined the DFN-AAI, gitlab.awi.de needs your organization to deliver information about your institutional e-mail address as part of the identity provided. Please contact the person responsible for shibboleth in your organization.

15.2 ESM Runscripts

1. **Q**: I get the error: `load_all_functions:  not found [No such file or directory]` when calling my runscript like this:

   ```
   $ ./my_run_script.sh -e some_expid
   ```

   A: You are trying to call your runscript the old-fashioned way that worked with the shell-script version, until revision 3. With the new python version, you get a new executable `esm_runscripts` that should be in your PATH already. Call your runscript like this:

   ```
   $ esm_runscripts my_run_script.sh -e some_expid
   ```

   All the command line options still apply. By the way, "load_all_function" doesn't hurt to have in the runscript, but can savely be removed.

2. **Q**: What should I put into the variable `FUNCTION_PATH` in my runscript, I can't find the folder `functions/all` it should point to.

   A: You can safely forget about `FUNCTION_PATH`, which was only needed in the shell script version until revision 3. Either ignore it, or better remove it from the runscript.

3. **Q**: When I try to branch-off from a spinup experiment using *FESOM*, I get the following runtime error:

   ```
   read ocean restart file
   Error:
   NetCDF: Invalid dimension ID or name
   ```

A: See How to branch-off FESOM from old spinup restart files.

15.3 ESM Master

1. **Q**: How can I define different environments for different models / different versions of the same model?

   A: You can add a choose-block in the models yaml-file (`esm_tools/configs/model_name.yaml`), e.g.:

```
choose_version:
        40r1:
                environment_changes:
                        add_export_vars:
                                - 'MY_VAR="something"'
                        add_module_actions:
                                - load my_own_module

        43r3:
                environment_changes:
                        add_export_vars:
                                - 'MY_VAR="something_else"'
```

2. **Q**: How can I add a new model, setup, and coupling strategy to the esm_master tool?

   A: Add your configuration in the file configs/esm_master/setups2models.yaml

15.4 Frequent Errors

1. **Q**: When I try to install *ESM-Tools* or use `esm_versions` I get the following error:

```
RuntimeError: Click will abort further execution because Python 3 was configured␣
↪to use ASCII as encoding for the environment. Consult https://click.
↪palletsprojects.com/en/7.x/python3/ for mitigation steps.
```

or something on the following lines:

```
 ERROR: Command errored out with exit status 1:
 command: /sw/rhel6-x64/conda/anaconda3-bleeding_edge/bin/python -c 'import sys,␣
↪setuptools, tokenize; sys.argv[0] = '"'"'/tmp/pip-install-0y687gmq/esm-master/setup.
↪py'"'"'; _file__='"'"'/tmp/pip-install-0y687gmq/esm-master/setup.py'"'"';
↪f=getattr(tokenize, '"'"'open'"'"', open)(__file__);code=f.read().replace('"'"'\r\n'
↪"'"', '"'"'\n'"'"');f.close();exec(compile(code, _file__, '"'"'exec'"'"'))' egg_
↪info --egg-base /tmp/pip-install-0y687gmq/esm-master/pip-egg-info
 cwd: /tmp/pip-install-0y687gmq/esm-master/
 Complete output (7 lines):
 Traceback (most recent call last):
   File "<string>", line 1, in <module>
   File "/tmp/pip-install-0y687gmq/esm-master/setup.py", line 8, in <module>
     readme = readme_file.read()
   File "/sw/rhel6-x64/conda/anaconda3-bleeding_edge/lib/python3.6/encodings/ascii.py
↪", line 26, in decode
     return codecs.ascii_decode(input, self.errors)[0]
 UnicodeDecodeError: 'ascii' codec can't decode byte 0xf0 in position 1468: ordinal␣
↪not in range(128)
     ----------------------------------------
```

(continues on next page)

(continued from previous page)

```
 ERROR: Command errored out with exit status 1: python setup.py egg_info Check the␣
↪logs for full command output.

**A**: Some systems have ``C.UTF-8`` as locale default (i.e. ``$LC_ALL``, ``$LANG``).␣
↪This issue is solved by setting up the locales respectively to ``en_US.UTF-8`` and␣
↪``en_US.UTF-8``, either manually or adding them to the local bash configuration␣
↪file (i.e. ``~/.bash_profile``)::

    $ export LC_ALL=en_US.UTF-8
    $ export LANG=en_US.UTF-8
```

2. **Q**: How can I add a new model, setup, and coupling strategy to the esm_master tool?

   A: Add your configuration in the file `configs/esm_master/setups2models.yaml` (see contributing:Implementing a New Model and *Implement a New Coupled Setup*)

CHAPTER

SIXTEEN

PYTHON PACKAGES

The ESM-Tools are divided into a number of python packages / git repositories, both to ensure stability of the code as well as reusability:

16.1 esm_tools.git

The only repository to clone by hand by the user, `esm_tools.git` contains the subfolders

configs: A collection of yaml configuration files, containing all the information needed by the python packages to work properly. This includes machine specific files (e.g. `machines/mistral.yaml`) , model specific files (e.g `fesom/fesom-2.0.yaml`), configurations for coupled setups (e.g. `foci/foci.yaml`), but also files with the information on how a certain software works (`batch_systems/slurm.yaml`), and finally, how the esm_tools themselves are supposed to work (e.g. `esm_master/esm_master.yaml`).

16.2 esm_master.git

This repository contains the python files that give the `esm_master` executable in the subfolder `esm_master`.

16.3 esm_runscripts.git

The python package of the `esm_runscripts` executable. The main routines can be found in `esm_runscripts/esm_sim_objects.py`.

16.4 esm_parser.git

In order to provide the additional functionality to the `yaml+` configuration files (like choose blocks, simple math operations, variable expansions etc.). `esm_parser` is an extension of the pyyaml package, it needs the `esm_calendar` package to run, but can otherwise easily be used to add `yaml+` configurations to any python software.

16.5 esm_calendar.git

CHAPTER

SEVENTEEN

ESM TOOLS CODE DOCUMENTATION

17.1 esm_archiving package

Top-level package for ESM Archiving.

esm_archiving.**archive_mistral**(*tfile*, *rtfile=None*)
Puts the `tfile` to the tape archive using `tape_command`

Parameters

- **tfile** (*str*) – The full path of the file to put to tape
- **rtfile** (*str*) – The filename on the remote tape server. Defaults to None, in which case a replacement is performed to keep as much of the filename the same as possible. Example: /work/ab0246/a270077/experiment.tgz –> /hpss/arch/ab0246/a270077/experiment.tgz

Returns

Return type None

esm_archiving.**check_tar_lists**(*tar_lists*)

esm_archiving.**delete_original_data**(*tfile*, *force=False*)
Erases data which is found in the tar file.

Parameters

- **tfile** (*str*) – Path to the tarfille whose data should be erased.
- **force** (*bool*) – If False, asks the user if they really want to delete their files. Otherwise just does this silently. Default is `False`

Returns

Return type None

esm_archiving.**determine_datestamp_location**(*files*)
Given a list of files; figures where the datestamp is by checking if it varies.

Parameters **files** (*list*) – A list (longer than 1!) of files to check

Returns A slice object giving the location of the datestamp

Return type slice

Raises **DatestampLocationError** : – Raised if there is more than one slice found where the numbers vary over different files -or- if the length of the file list is not longer than 1.

esm_archiving.**determine_potential_datestamp_locations**(*filepattern*)
For a filepattern, gives back index of potential date locations

Parameters **filepattern** (*str*) – The filepattern to check.

Returns A list of slice object which you can use to cut out dates from the filepattern

Return type list

esm_archiving.**find_indices_of**(*char*, *in_string*)
Finds indicies of a specific character in a string

Parameters

- **char** (*str*) – The character to look for
- **in_string** (*str*) – The string to look in

Yields *int* – Each round of the generator gives you the next index for the desired character.

esm_archiving.**get_files_for_date_range**(*filepattern*, *start_date*, *stop_date*, *frequency*, *date_format='%Y%m%d'*)
Creates a list of files for specified start/stop dates

Parameters

- **filepattern** (*str*) – A filepattern to replace dates in
- **start_date** (*str*) – The starting date, in a pandas-friendly date format
- **stop_date** (*str*) – Ending date, pandas friendly. Note that for end dates, you need to **add one month** to assure that you get the last step in your list!
- **frequency** (*str*) – Frequency of dates, pandas friendly
- **date_format** (*str*) – How dates should be formatted, defaults to %Y%m%d

Returns A list of strings for the filepattern with correct date stamps.

Return type list

Example

```
>>> filepattern =  "LGM_24hourly_PMIP4_echam6_BOT_mm_>>>DATE<<<.nc"
>>> get_files_for_date_range(filepattern, "1890-07", "1891-11", "1M", date_format=
↪"%Y%m")
[
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189007.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189008.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189009.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189010.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189011.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189012.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189101.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189102.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189103.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189104.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189105.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189106.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189107.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189108.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189109.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189110.nc",
]
```

esm_archiving.**get_list_from_filepattern**(*filepattern*)

esm_archiving.**group_files**(*top*, *filetype*)
Generates quasi-regexes for a specific filetype, replacing all numbers with #.

Parameters

- **top** (*str*) – Where to start looking (this should normally be top of the experiment)
- **filetype** (*str*) – Which files to go through (e.g. outdata, restart, etc...)

Returns A dictonary containing keys for each folder found in `filetype`, and values as lists of files with strings where numbers are replaced by #.

Return type dict

esm_archiving.**group_indexes**(*index_list*)
Splits indexes into tuples of monotonically ascending values.

Parameters **list** – The list to split up

Returns A list of tuples, so that you can get only one group of ascending tuples.

Return type list

Example

```
>>> indexes = [0, 1, 2, 3, 12, 13, 15, 16]
>>> group_indexes(indexes)
[(0, 1, 2, 3), (12, 13), (15, 16)]
```

esm_archiving.**log_tarfile_contents**(*tfile*)
Generates a log of the tarball contents

Parameters **tfile** (*str*) – The path for the tar file to generate a log for

Returns

Return type None

Warning: Note that for this function to work, you need to have write permission in the directory where the tarball is located. If not, this will probably raise an OSError. I can imagine giving the location of the log path as an argument; but would like to see if that is actually needed before implementing it...

esm_archiving.**pack_tarfile**(*flist*, *wdir*, *outname*)
Creates a compressed tarball (`outname`) with all files found in `flist`.

Parameters

- **flist** (*list*) – A list of files to include in this tarball
- **wdir** (*str*) – The directory to "change" to when packing up the tar file. This will (essentially) be used in the tar command as the -C option by stripping off the beginning of the flist
- **outname** (*str*) – The output file name

Returns The output file name

Return type str

esm_archiving.**purify_expid_in**(*model_files*, *expid*, *restore=False*)
Puts or restores >>>EXPID<<< marker in filepatterns

Parameters

- **model_files** (*dict*) – The model files for archiving
- **expid** (*str*) – The experiment ID to purify or restore
- **restore** (*bool*) – Set experiment ID back from the temporary marker

Returns Dictionary containing keys for each model, values for file patterns

Return type dict

esm_archiving.**sort_files_to_tarlists**(*model_files*, *start_date*, *end_date*, *config*)

esm_archiving.**split_list_due_to_size_limit**(*in_list*, *slimit*)

esm_archiving.**stamp_filepattern**(*filepattern*, *force_return=False*)

Transforms # in filepatterns to >>>DATE<<< and replaces other numbers back to original

Parameters

- **filepattern** (*str*) – Filepattern to get date stamps for
- **force_return** (*bool*) – Returns the list of filepatterns even if it is longer than 1.

Returns New filepattern, with >>>DATE<<<

Return type str

esm_archiving.**stamp_files**(*model_files*)

Given a sttandard file dictioanry (keys: model names, values: filepattern); figures out where the date probably is, and replaces the # sequence with a >>>DATE<<< stamp.

Parameters **model_files** (*dict*) – Dictionary of keys (model names) where values are lists of files for each model.

Returns As the input, but replaces the filepatterns with the >>>DATE<<< stamp.

Return type dict

esm_archiving.**sum_tar_lists**(*tar_lists*)

Sums up the amount of space in the tar lists dictionary

Given `tar_lists`, which is generally a dicitonary consisting of keys (model names) and values (files to be tarred), figures out how much space the **raw, uncompressed** files would use. Generally the compressed tarball will take up less space.

Parameters **tar_lists** (*dict*) – Dictionary of file lists to be summed up. Reports every sum as a value for the key of that particular list.

Returns Keys are the same as in the input, values are the sums (in bytes) of all files present within the list.

Return type dict

esm_archiving.**sum_tar_lists_human_readable**(*tar_lists*)

As `sum_tar_lists` but gives back strings with human-readable sizes.

17.1.1 Subpackages

esm_archiving.database package

The database module for archiving.

The database extension allows you keep track of which experiments are on the tape, which files are in which tarball, along with some experiment meta-data.

Submodules

esm_archiving.database.model module

The database module for archiving.

The database extension allows you keep track of which experiments are on the tape, which files are in which tarball, along with some experiment meta-data.

class esm_archiving.database.model.**Archive**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

exp_ref

expid_id

id

tarballs

class esm_archiving.database.model.**ArchivedFile**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

fname

id

on_disk

on_tape

tarball

tarball_id

class esm_archiving.database.model.**Experiments**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

archive

created_at

expid

id

class esm_archiving.database.model.**Tarball**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

archive

archive_id

files

fname

id

esm_archiving.database.utils module

esm_archiving.external package

Submodules

esm_archiving.external.pypftp module

class esm_archiving.external.pypftp.**Pftp**(*username=None*, *password=None*)
Bases: object

HOST = 'tape.dkrz.de'

PORT = 4021

close()

cwd(*path*)
change working directory

directories(*path=None*)
gather directories at the given path

static download(*source*, *destination*)
uses pftp binary for transfering the file

exists(*path*)
check if a path exists

files(*path=None*)
gather files at the given path

is_connected()
check if the connection is still active

isdir(*pathname*)
Returns true if pathname refers to an existing directory

isfile(*pathname*)
Returns true if pathname refers to an existing file

islink(*pathname*)

listdir(*path=None*)
list directory contents

listing(*path=None*)
list directory contents

listing2(*path=None*)
directory listing in long form. similar to "ls -l"

makedirs(*path*)
Recursively create dirs as required walking up to an existing parent dir

mkdir(*path*)

mlsd(*path*)

pwd()
present working directory

quit()

reconnect()
reconnects to the ftp server

remove(*filename*)

removedirs(*path*)

rename(*from_name*, *to_name*)

rmdir(*path*)
remove directory

size(*pathname*)
Returns size of path in bytes

stat(*pathname*)
Returns stat of the path

static upload(*source*, *destination*)
uses pftp binary for transfering the file

walk(*path=None*)
recursively walk the directory tree from the given path. Similar to os.walk

walk_for_directories(*path=None*)
recursively gather directories

walk_for_files(*path=None*)
recursively gather files

esm_archiving.external.pypftp.**download**(*source*, *destination*)

esm_archiving.external.pypftp.**upload**(*source*, *destination*)

17.1.2 Submodules

17.1.3 esm_archiving.cli module

After installation, you have a new command in your path:

```
esm_archive
```

Passing in the argument `--help` will show available subcommands:

```
Usage: esm_archive [OPTIONS] COMMAND [ARGS]...

  Console script for esm_archiving.

Options:
  --version             Show the version and exit.
  --write_local_config  Write a local configuration YAML file in the current
                        working directory
  --write_config        Write a global configuration YAML file in
                        ~/.config/esm_archiving/
  --help                Show this message and exit.
```

(continues on next page)

(continued from previous page)

```
Commands:
  create
  upload
```

To use the tool, you can first `create` a tar archive and then use `upload` to put it onto the tape server.

Creating tarballs

Use `esm_archive create` to generate tar files from an experiment:

```
esm_archive create /path/to/top/of/experiment start_date end_date
```

The arguments `start_date` and `end_date` should take the form `YYYY-MM-DD`. A complete example would be:

```
esm_archive create /work/ab0246/a270077/from_ba0989/AWICM/LGM_6hours 1850-01-01 1851-
↪01-01
```

The archiving tool will automatically pack up all files it finds matching these dates in the `outdata` and `restart` directories and generate logs in the top of the experiment folder. Note that the final date (1851-01-1 in this example) is **not included**. During packing, you get a progress bar indicating when the tarball is finished.

Please be aware that are size limits in place on DKRZ's tape server. Any tar files **larger than 500 Gb will be trucated**. For more information, see: https://www.dkrz.de/up/systems/hpss/hpss

Uploading tarballs

A second command `esm_archive upload` allows you to put tarballs onto to tape server at DKRZ:

```
esm_archive upload /path/to/top/of/experiment start_date end_date
```

The signature is the same as for the `create` subcommand. Note that for this to work; you need to have a properly configured `.netrc` file in your home directory:

```
$ cat ~/.netrc
machine tape.dkrz.de login a270077 password OMITTED
```

This file needs to be readable/writable **only** for you, e.g. `chmod 600`. The archiving program will then be able to automatically log into the tape server and upload the tarballs. Again, more information about logging onto the tape server without password authentication can be found here: https://www.dkrz.de/up/help/faq/hpss/how-can-i-use-the-hpss-tape-archive-without-typing-my-password-every-time-e-g-in-scripts-or-jobs

17.1.4 esm_archiving.config module

When run from either the command line or in library mode (note **not** as an ESM Plugin), `esm_archiving` can be configured to how it looks for specific files. The configuration file is called `esm_archiving_config`, should be written in YAML, and have the following format:

```
echam:  # The model name
    archive: # archive seperator **required**
        # Frequency specification (how often
        # a datestamp is generated to look for)
```

(continues on next page)

(continued from previous page)

```
        frequency: "1M"
        # Date format specification
        date_format: "%Y%m"
```

By default, `esm_archive` looks in the following locations:

1. Current working directory
2. **Any files in the XDG Standard:** https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html

If nothing is found, the program reverts to the hard-coded defaults, found in `esm_archiving/esm_archiving/config.py`

Note: In future, it might be changed that the program will look for an experiment specific configuration based upon the path it is given during the `create` or `upload` step.

Generating a configuration

You can use the command line switches `--write_local_config` and `--write_config` to generate configuration files either in the current working directory, or in the global directory for your user account defined by the XDG standard (typically ~/.config/esm_archiving):

```
$ esm_archive --write_local_config
Writing local (experiment) configuration...

$ esm_archive --write_config
Writing global (user) configuration...
```

esm_archiving.config.**load_config**()
Loads the configuration from one of the default configuration directories. If none can be found, returns the hard-coded default configuration.

Returns A representation of the configuration used for archiving.

Return type dict

esm_archiving.config.**write_config_yaml**(*path=None*)

17.1.5 esm_archiving.esm_archiving module

This is the `esm_archiving` module.

exception esm_archiving.esm_archiving.**DatestampLocationError**
Bases: `Exception`

esm_archiving.esm_archiving.**archive_mistral**(*tfile*, *rtfile=None*)
Puts the `tfile` to the tape archive using `tape_command`

Parameters

- **tfile** (*str*) – The full path of the file to put to tape
- **rtfile** (*str*) – The filename on the remote tape server. Defaults to None, in which case a replacement is performed to keep as much of the filename the same as possible. Example: /work/ab0246/a270077/experiment.tgz –> /hpss/arch/ab0246/a270077/experiment.tgz

Returns

Return type None

esm_archiving.esm_archiving.**check_tar_lists**(*tar_lists*)

esm_archiving.esm_archiving.**delete_original_data**(*tfile*, *force=False*)
Erases data which is found in the tar file.

Parameters

- **tfile** (*str*) – Path to the tarfille whose data should be erased.
- **force** (*bool*) – If False, asks the user if they really want to delete their files. Otherwise just does this silently. Default is `False`

Returns

Return type None

esm_archiving.esm_archiving.**determine_datestamp_location**(*files*)
Given a list of files; figures where the datestamp is by checking if it varies.

Parameters **files** (*list*) – A list (longer than 1!) of files to check

Returns A slice object giving the location of the datestamp

Return type slice

Raises **DatestampLocationError :** – Raised if there is more than one slice found where the numbers vary over different files -or- if the length of the file list is not longer than 1.

esm_archiving.esm_archiving.**determine_potential_datestamp_locations**(*filepattern*)
For a filepattern, gives back index of potential date locations

Parameters **filepattern** (*str*) – The filepattern to check.

Returns A list of slice object which you can use to cut out dates from the filepattern

Return type list

esm_archiving.esm_archiving.**find_indices_of**(*char*, *in_string*)
Finds indicies of a specific character in a string

Parameters

- **char** (*str*) – The character to look for
- **in_string** (*str*) – The string to look in

Yields *int* – Each round of the generator gives you the next index for the desired character.

esm_archiving.esm_archiving.**get_files_for_date_range**(*filepattern*, *start_date*, *stop_date*, *frequency*, *date_format='%Y%m%d'*)
Creates a list of files for specified start/stop dates

Parameters

- **filepattern** (*str*) – A filepattern to replace dates in
- **start_date** (*str*) – The starting date, in a pandas-friendly date format
- **stop_date** (*str*) – Ending date, pandas friendly. Note that for end dates, you need to **add one month** to assure that you get the last step in your list!
- **frequency** (*str*) – Frequency of dates, pandas friendly
- **date_format** (*str*) – How dates should be formatted, defaults to %Y%m%d

Returns A list of strings for the filepattern with correct date stamps.

Return type list

Example

```
>>> filepattern =  "LGM_24hourly_PMIP4_echam6_BOT_mm_>>>DATE<<<.nc"
>>> get_files_for_date_range(filepattern, "1890-07", "1891-11", "1M", date_format=
↪"%Y%m")
[
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189007.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189008.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189009.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189010.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189011.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189012.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189101.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189102.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189103.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189104.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189105.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189106.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189107.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189108.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189109.nc",
    "LGM_24hourly_PMIP4_echam6_BOT_mm_189110.nc",
]
```

esm_archiving.esm_archiving.**get_list_from_filepattern**(*filepattern*)

esm_archiving.esm_archiving.**group_files**(*top*, *filetype*)

Generates quasi-regexes for a specific filetype, replacing all numbers with #.

Parameters

- **top** (*str*) – Where to start looking (this should normally be top of the experiment)
- **filetype** (*str*) – Which files to go through (e.g. outdata, restart, etc...)

Returns A dictonary containing keys for each folder found in `filetype`, and values as lists of files with strings where numbers are replaced by #.

Return type dict

esm_archiving.esm_archiving.**group_indexes**(*index_list*)

Splits indexes into tuples of monotonically ascending values.

Parameters **list** – The list to split up

Returns A list of tuples, so that you can get only one group of ascending tuples.

Return type list

Example

```
>>> indexes = [0, 1, 2, 3, 12, 13, 15, 16]
>>> group_indexes(indexes)
[(0, 1, 2, 3), (12, 13), (15, 16)]
```

esm_archiving.esm_archiving.**log_tarfile_contents**(*tfile*)
Generates a log of the tarball contents

Parameters **tfile** (*str*) – The path for the tar file to generate a log for

Returns

Return type None

Warning: Note that for this function to work, you need to have write permission in the directory where the tarball is located. If not, this will probably raise an OSError. I can imagine giving the location of the log path as an argument; but would like to see if that is actually needed before implementing it...

esm_archiving.esm_archiving.**pack_tarfile**(*flist*, *wdir*, *outname*)
Creates a compressed tarball (`outname`) with all files found in `flist`.

Parameters

- **flist** (*list*) – A list of files to include in this tarball
- **wdir** (*str*) – The directory to "change" to when packing up the tar file. This will (essentially) be used in the tar command as the -C option by stripping off the beginning of the flist
- **outname** (*str*) – The output file name

Returns The output file name

Return type str

esm_archiving.esm_archiving.**purify_expid_in**(*model_files*, *expid*, *restore=False*)
Puts or restores >>>EXPID<<< marker in filepatterns

Parameters

- **model_files** (*dict*) – The model files for archiving
- **expid** (*str*) – The experiment ID to purify or restore
- **restore** (*bool*) – Set experiment ID back from the temporary marker

Returns Dictionary containing keys for each model, values for file patterns

Return type dict

esm_archiving.esm_archiving.**query_yes_no**(*question*, *default='yes'*)
Ask a yes/no question via `input()` and return their answer.

"question" is a string that is presented to the user. "default" is the presumed answer if the user just hits <Enter>.

It must be "yes" (the default), "no" or None (meaning an answer is required of the user).

The "answer" return value is True for "yes" or False for "no".

Note: Shamelessly stolen from StackOverflow It's not hard to implement, but Paul is lazy...

Parameters

- **question** (*str*) – The question you'd like to ask the user
- **default** (*str*) – The presumed answer for `question`. Defaults to "yes".

Returns True if the user said yes, False if the use said no.

Return type bool

esm_archiving.esm_archiving.**run_command**(*command*)
Runs `command` and directly prints output to screen.

Parameters **command** (*str*) – The command to run, with pipes, redirects, whatever

Returns **rc** – The return code of the subprocess.

Return type int

esm_archiving.esm_archiving.**sort_files_to_tarlists**(*model_files*, *start_date*, *end_date*, *config*)

esm_archiving.esm_archiving.**split_list_due_to_size_limit**(*in_list*, *slimit*)

esm_archiving.esm_archiving.**stamp_filepattern**(*filepattern*, *force_return=False*)
Transforms # in filepatterns to >>>DATE<<< and replaces other numbers back to original

Parameters

- **filepattern** (*str*) – Filepattern to get date stamps for
- **force_return** (*bool*) – Returns the list of filepatterns even if it is longer than 1.

Returns New filepattern, with >>>DATE<<<

Return type str

esm_archiving.esm_archiving.**stamp_files**(*model_files*)
Given a sttandard file dictioanry (keys: model names, values: filepattern); figures out where the date probably is, and replaces the # sequence with a >>>DATE<<< stamp.

Parameters **model_files** (*dict*) – Dictionary of keys (model names) where values are lists of files for each model.

Returns As the input, but replaces the filepatterns with the >>>DATE<<< stamp.

Return type dict

esm_archiving.esm_archiving.**sum_tar_lists**(*tar_lists*)
Sums up the amount of space in the tar lists dictionary

Given `tar_lists`, which is generally a dicitonary consisting of keys (model names) and values (files to be tarred), figures out how much space the **raw, uncompressed** files would use. Generally the compressed tarball will take up less space.

Parameters **tar_lists** (*dict*) – Dictionary of file lists to be summed up. Reports every sum as a value for the key of that particular list.

Returns Keys are the same as in the input, values are the sums (in bytes) of all files present within the list.

Return type dict

esm_archiving.esm_archiving.**sum_tar_lists_human_readable**(*tar_lists*)
As `sum_tar_lists` but gives back strings with human-readable sizes.

17.2 esm_calendar package

Top-level package for ESM Calendar.

17.2.1 Submodules

17.2.2 esm_calendar.esm_calendar module

Module Docstring.,..?

class esm_calendar.esm_calendar.**Calendar**(*calendar_type=1*)
Bases: object

Class to contain various types of calendars.

Parameters **calendar_type** (*int*) – The type of calendar to use.

Supported calendar types: 0

no leap years

1 proleptic greogrian calendar (default)

n equal months of n days

timeunits
A list of accepted time units.

Type list of str

monthnames
A list of valid month names, using 3 letter English abbreviation.

Type list of str

isleapyear(*year*)
Returns a boolean testing if the given year is a leapyear

day_in_year(year):
Returns the total number of days in a given year

day_in_month(year, month):
Returns the total number of days in a given month for a given year (considering leapyears)

day_in_month(*year*, *month*)
Finds the number of days in a given month

Parameters

- **year** (*int*) – The year to check
- **month** (*int or str*) – The month number or short name.

Returns The number of days in this month, considering leapyears if needed.

Return type int

Raises **TypeError** – Raised when you give an incorrect type for month

day_in_year(*year*)
Finds total number of days in a year, considering leapyears if the calendar type allows for them.

Parameters **year** (*int*) – The year to check

Returns The total number of days for this specific calendar type

Return type int

isleapyear(*year*)
Checks if a year is a leapyear

Parameters **year** (*int*) – The year to check

Returns True if the given year is a leapyear

Return type bool

monthnames = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'N

timeunits = ['years', 'months', 'days', 'hours', 'minutes', 'seconds']

class esm_calendar.esm_calendar.**Date**(*indate*, *calendar=esm_calendar(calendar_type=1)*)
Bases: object

A class to contain dates, also compatiable with paleo (negative dates)

Parameters

- **indate** (*str*) – The date to use.

  See *pyesm.core.time_control.esm_calendar.Dateformat* for available formatters.

- **calendar** (*Calendar`, optional*) – The type of calendar to use. Defaults to a greogrian proleptic calendar if nothing is specified.

year
The year

Type int

month
The month

Type int

day
The day

Type int

hour
The hour

Type int

minute
The minute

Type int

second
The second

Type int

_calendar
The type of calendar to use

Type Calendar`

add(*to_add*)
Adds another date to this one.

Parameters **to_add** (*Date`*) – The other date to add to this one.

Returns **new_date** – A new date object with the added dates

Return type Date`

day_of_year()
Gets the day of the year, counting from Jan. 1

Returns The day of the current year.

Return type int

format(*form='SELF'*, *givenph=None*, *givenpm=None*, *givenps=None*)
Needs a docstring! The following forms are accepted: + SELF: uses the format which was given when constructing the date + 0: A Date formated as YYYY

In [5]: test.format(form=1) Out[5]: '1850-01-01_00:00:00'

In [6]: test.format(form=2) Out[6]: '1850-01-01T00:00:00'

In [7]: test.format(form=3) Out[7]: '1850-01-01 00:00:00'

In [8]: test.format(form=4) Out[8]: '1850 01 01 00 00 00'

In [9]: test.format(form=5) Out[9]: '01 Jan 1850 00:00:00'

In [10]: test.format(form=6) Out[10]: '18500101_00:00:00'

In [11]: test.format(form=7) Out[11]: '1850-01-01_000000'

In [12]: test.format(form=8) Out[12]: '18500101000000'

In [13]: test.format(form=9) Out[13]: '18500101_000000'

In [14]: test.format(form=10) Out[14]: '01/01/1850 00:00:00'

classmethod from_list(*_list*)
Creates a new Date from a list

Parameters **_list** (*list of ints*) – A list of [year, month, day, hour, minute, second]

Returns **date** – A new date of year month day, hour minute, second

Return type Date`

classmethod fromlist(*_list*)
Creates a new Date from a list

Parameters **_list** (*list of ints*) – A list of [year, month, day, hour, minute, second]

Returns **date** – A new date of year month day, hour minute, second

Return type Date`

makesense(*ndate*)
Puts overflowed time back into the correct unit.

When manipulating the date, it might be that you have "70 seconds", or something similar. Here, we put the overflowed time into the appropriate unit.

output(*form='SELF'*)

property sday

property sdoy

property shour

property sminute

property smonth

property ssecond

sub_date(*other*)

sub_tuple(*to_sub*)
Adds another date to from one.

Parameters **to_sub** (*Date`*) – The other date to sub from this one.

Returns **new_date** – A new date object with the subtracted dates

Return type Date`

property syear

time_between(*date*, *outformat='seconds'*)
Computes the time between two dates

Parameters **date** (*date`*) – The date to compare against.

Returns

Return type

??

class esm_calendar.esm_calendar.**Dateformat**(*form=1*, *printhours=True*, *printminutes=True*, *printseconds=True*)
Bases: object

datesep = ['', '-', '-', '-', ' ', ' ', '', '-', '', '', '/']

dtsep = ['_', '_', 'T', ' ', ' ', ' ', '_', '_', '', '_', ' ']

timesep = ['', ':', ':', ':', ' ', ':', ':', '', '', '', ':']

esm_calendar.esm_calendar.**find_remaining_hours**(*seconds*)
Finds the remaining full minutes of a given number of seconds

Parameters **seconds** (*int*) – The number of seconds to allocate

Returns The leftover seconds once new minutes have been filled.

Return type int

esm_calendar.esm_calendar.**find_remaining_minutes**(*seconds*)
Finds the remaining full minutes of a given number of seconds

Parameters **seconds** (*int*) – The number of seconds to allocate

Returns The leftover seconds once new minutes have been filled.

Return type int

17.3 esm_database package

Top-level package for ESM Database.

17.3.1 Submodules

17.3.2 esm_database.cli module

A small wrapper that combines the shell interface and the Python interface

esm_database.cli.**main**()

esm_database.cli.**parse_shargs**()
The arg parser for interactive use

17.3.3 esm_database.esm_database module

class esm_database.esm_database.**DisplayDatabase**(*tablename=None*)
Bases: object

ask_column()

ask_dataset()

decision_maker()

edit_dataset()

output_writer()

remove_datasets()

select_stuff()

17.3.4 esm_database.getch module

esm_database.getch.**get_one_of**(*testlist*)

17.3.5 esm_database.location_database module

class esm_database.location_database.**database_location**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

class_in

id

location

table_name

static topline()

esm_database.location_database.**register**(*table_name*, *given_location*, *class_in*)

17.4 esm_environment package

Top-level package for ESM Environment.

17.4.1 Submodules

17.4.2 esm_environment.esm_environment module

Main module for EsmEnvironment.

class esm_environment.esm_environment.**EnvironmentInfos**(*run_or_compile*, *complete_config=None*, *model=None*)

Bases: object

static add_commands(*commands*, *name*)
Writes all commands in a list to a file named <name>_script.sh, located in the current working directory. The header from this script is read from dummy_script.sh, also in the current working directory.

Parameters

- **commands** (*list of str*) – List of the commands to write to the file after the header
- **name** (*str*) – Name of the script, generally something like comp_echam-6.3.05

Returns name + "_script.sh"

Return type str

add_esm_var()
Adds the ENVIRONMENT_SET_BY_ESMTOOLS=TRUE to the config, for later dumping to the shell script.

apply_config_changes(*run_or_compile*, *config*, *model*)

apply_model_changes(*model*, *run_or_compile='runtime'*, *modelconfig=None*)

static cleanup_dummy_script()
Removes the dummy_script.sh if it exists.

get_shell_commands()
Gathers module actions and export variables from the config to a list, prepending appropriate shell command words (e.g. module and export)

Returns

Return type list

output()

replace_model_dir(*model_dir*)
Replaces any instances of ${model_dir} in the config section "export_vars" with the argument

Parameters **model_dir** (*str*) – The replacement string for ${model_dir}

write_dummy_script(*include_set_e=True*)
Writes a dummy script containing only the header information, module commands, and export variables. The actual compile/configure commands are added later.

Parameters **include_set_e** (*bool*) – Default to True, whether or not to include a set -e at the beginning of the script. This causes the shell to stop as soon as an error is encountered.

class esm_environment.esm_environment.**environment_infos**(**args*, ***kwargs*)
Bases: *esm_environment.esm_environment.EnvironmentInfos*

17.5 esm_master package

Top-level package for ESM Master.

17.5.1 Submodules

17.5.2 esm_master.cli module

Console script for esm_master.

esm_master.cli.**main**()

17.5.3 esm_master.compile_info module

17.5.4 esm_master.database module

class esm_master.database.**installation**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

action

folder

id

static nicer_output(*run*)

setup_name

timestamp

static topline()

17.5.5 esm_master.database_actions module

esm_master.database_actions.**database_entry**(*action*, *setup_name*, *base_dir*)

17.5.6 esm_master.esm_master module

17.5.7 esm_master.general_stuff module

17.5.8 esm_master.software_package module

esm_master.software_package.**replace_var**(*var*, *tag*, *value*)

class esm_master.software_package.**software_package**(*raw*, *setup_info*, *vcs*, *general*, *no_infos=False*)
Bases: object

complete_targets(*setup_info*)

fill_in_infos(*setup_info*, *vcs*, *general*)

get_command_list(*setup_info*, *vcs*, *general*)

get_comp_type(*setup_info*)

get_coupling_changes(*setup_info*)

get_repo_info(*setup_info*, *vcs*)

get_subpackages(*setup_info*, *vcs*, *general*)

get_targets(*setup_info*, *vcs*)

output()

17.5.9 esm_master.task module

class esm_master.task.**Task**(*raw*, *setup_info*, *vcs*, *general*, *complete_config*)

Bases: object

What you can do with a software_package, e.g. comp-awicm-2.0

assemble_command_list()

check_if_download_task(*setup_info*)

check_if_target(*setup_info*)

check_requirements()

cleanup_script()

compile_binaries()

download_folders()

execute()

get_subtasks(*setup_info*, *vcs*, *general*, *complete_config*)

list_required_dirs()

order_subtasks(*setup_info*, *vcs*, *general*)

output()

output_steps()

validate()

validate_only_subtask()

esm_master.task.**install**(*package*)

17.6 esm_parser package

Top-level package for ESM Parser.

17.6.1 Submodules

17.6.2 esm_parser.esm_parser module

YAML Parser for Earth System Models

One core element of the `esm-tools` is the description of model configurations and experiments with the aid of `YAML` files. Beyond the standard features of `YAML`, several specific conventions have been implemented to ease the description of your simulations. These conventions are described below, and the functions which implement them are documented with minimal examples. Internally, after parsing the `YAML` files are converted into a single Python dictionary.

Parsing takes place by initializing objects which represent either an entire setup, `ConfigSetup`, or a specific component, `ConfigComponent`. Both of these objects base off of `GeneralConfig`, which is a dictionary subclass performing specific parsing steps during the object's creation. The parsing steps are presented in the order that they are resolved:

When initializing a `ConfigSetup` or `ConfigComponent`, a name of the desired setup or component must be given, e.g. `"awicm"` or `"echam"`. This configuration is immediately loaded along with any further configs listed in the section "further_reading". Note that this means that **any configuration listed in "further_reading" must not contain any variables!!**

Following this step, a method called `_config_init` is run for all classes based off of `GeneralConfig`. For components, any entries listed under `"include_submodels"` are attached and registed under a new keyword `"submodels"`.

For setups, the next step is to determine the computing host and load the appropriate configuration files. Setups divide their configuration into 3 specific parts:

1. Setup information, contained under `config['setup']`. This includes, e.g. information regarding a standalone setup, possible coupling, etc.
2. Model Information, under `config['model']`. This contains specific information for all models and submodels, such as resolution, input file names, namelists, etc.
3. User information, under `config['model']`. The user can specify to override any of the defaults with their own choices.

In the next step, all keys starting with `"choose_"` are determined, along with any information they set. This is done first for the setup, and then for the models. These are filtered to determine an independent choice, and if cyclic dependencies occur, an error is raised. All choices are then resolved until nothing is left.

Specific documentation for classes and functions are given below:

class esm_parser.esm_parser.**ConfigSetup**(*model*, *version*, *user_config*)

Bases: *esm_parser.esm_parser.GeneralConfig*

Config Class for Setups

finalize()

run_recursive_functions(*config*, *isblacklist=True*)

exception esm_parser.esm_parser.**EsmParserError**
Bases: Exception

Raise this error when the parser has problems

class esm_parser.esm_parser.**GeneralConfig**(*model*, *version*, *user_config*)
Bases: dict

All configs do this!

esm_parser.esm_parser.**actually_find_variable**(*tree*, *rhs*, *full_config*)

esm_parser.esm_parser.**add_entries_from_chapter**(*config*, *add_chapter*, *add_entries*)

esm_parser.esm_parser.**add_entries_to_chapter_in_config**(*model_config*, *valid_model_names*, *setup_config*, *valid_setup_names*)

esm_parser.esm_parser.**add_entry_to_chapter**(*add_chapter*, *add_entries*, *model_to_add_to*, *model_with_add_statement*, *model_config*, *setup_config*)

esm_parser.esm_parser.**add_more_important_tasks**(*choose_keyword*, *all_set_variables*, *task_list*)
Determines dependencies of a choose keyword.

Parameters

- **choose_keyword** (*str*) – The keyword, starting with choose, which is looked through to check if there are any dependencies that must be resolved first to correctly resolve this one.
- **all_set_variables** (*dict*) – All variables that can be set
- **task_list** (*list*) – A list in the order in which tasks must be resolved for choose_keyword to make sense.

Returns A list of choices which must be made in order for choose_keyword to make sense.

Return type task_list

esm_parser.esm_parser.**attach_single_config**(*config*, *path*, *attach_value*)

esm_parser.esm_parser.**attach_to_config_and_reduce_keyword**(*config_to_read_from*, *config_to_write_to*, *full_keyword*, *reduced_keyword='included_files'*, *level_to_write_to=None*)
Attaches a new dictionary to the config, and registers it as the value of reduced_keyword.

Parameters

- **config_to_read_from** (*dict*) – The configuration dictionary from which information is read from. The keyword from which additional YAML files are read from should be on the top level of this dictionary.
- **config_to_write_to** (*dict*) – The dictionary where the contents of config_to_read_from[full_keyword] is written in.
- **full_keyword** – The keyword where contents are extracted from
- **reduced_keyword** – The keyword where the contents of config_to_read_from[full_keyword] are written to

- **level_to_write_to** – If this is specified, the attached entries are written here instead of in the top level of config_to_write_to. Note that only one level down is currently supported.

The purpose behind this is to have a chapter in config "include_submodels" = ["echam", "fesom"], which would then find the "echam.yaml" and "fesom.yaml" configs, and attach them to "config" under config[submodels], and the entire config for e.g. echam would show up in config[echam

Since config_to_read_from and config_to_write_to are dict objects, they are modified **in place**. Note also that the entry config_to_read_from[full_keyword] is deleted at the end of the routine.

If the entry in config_to_read_from[full_keyword] is a list, each item in that list is split into two parts: model and model_part. For example:

```
>>> # Assuming: config_to_read_from[full_keyword] = ['echam.datasets', 'echam.
↪restart.streams']
>>> model, model_part = 'echam', 'datasets' # first part
>>> model, model_part = 'echam', 'restart.streams' # second part
```

The first part, in the example echam is used to determine where to look for new YAML files. Then, a yaml file corresponding to a file called echam.datasets.yaml is loaded, and attached to the config.

Warning: Both config_to_read_from and config_to_write_to are modified **in place**!

esm_parser.esm_parser.**attach_to_config_and_remove**(*config*, *attach_key*)

Attaches extra dict to this one and removes the chapter

Updates the dictionary on config with values from any file found under a listing specified by attach_key.

Parameters

- **config** (*dict*) – The configuration to update
- **attach_key** (*str*) – A key who's value points to a list of various yaml files to update config with.

Warning: The config is modified **in place**!

esm_parser.esm_parser.**basic_add_entries_to_chapter_in_config**(*config*)

esm_parser.esm_parser.**basic_add_more_important_tasks**(*choose_keyword*, *all_set_variables*, *task_list*)

Determines dependencies of a choose keyword.

Parameters

- **choose_keyword** (*str*) – The keyword, starting with choose, which is looked through to check if there are any dependencies that must be resolved first to correctly resolve this one.
- **all_set_variables** (*dict*) – All variables that can be set
- **task_list** (*list*) – A list in the order in which tasks must be resolved for choose_keyword to make sense.

Returns A list of choices which must be made in order for choose_keyword to make sense.

Return type task_list

esm_parser.esm_parser.**basic_choose_blocks**(*config_to_resolve*, *config_to_search*, *isblacklist=True*)

esm_parser.esm_parser.**basic_determine_set_variables_in_choose_block**(*config*)

esm_parser.esm_parser.**basic_find_add_entries_in_config**(*mapping*)

esm_parser.esm_parser.**basic_find_one_independent_choose**(*all_set_variables*)

Given a dictionary of `all_set_variables`, which comes out of the function `determine_set_variables_in_choose_block`, gives a list of task/variable dependencies to resolve in order to figure out the variable.

Parameters **all_set_variables** (*dict*) –

Returns **task_list** – A list of tuples comprising `(model_name, var_name)` in order to resolve one `choose_` block. This list is built in such a way that the beginning of the list provides dependencies for later on in the list.

Return type list

esm_parser.esm_parser.**basic_find_remove_entries_in_config**(*mapping*)

esm_parser.esm_parser.**basic_list_all_keys_starting_with_choose**(*mapping*, *ignore_list*, *isblacklist*)

esm_parser.esm_parser.**basic_remove_entries_from_chapter_in_config**(*config*)

esm_parser.esm_parser.**choose_blocks**(*config*, *blackdict={}*, *isblacklist=True*)

esm_parser.esm_parser.**complete_config**(*user_config*)

esm_parser.esm_parser.**convert**(*value*)

esm_parser.esm_parser.**could_be_bool**(*value*)

esm_parser.esm_parser.**could_be_complex**(*value*)

esm_parser.esm_parser.**could_be_float**(*value*)

esm_parser.esm_parser.**could_be_int**(*value*)

esm_parser.esm_parser.**deep_update**(*chapter*, *entries*, *config*, *blackdict={}*)

esm_parser.esm_parser.**del_value_for_nested_key**(*config*, *key*)

In a dict of dicts, delete a key/value pair.

Parameters

- **config** (*dict*) – The dict to delete in.
- **key** (*str*) – The key to delete.

> **Warning:** The `config` is modified **in place**!

esm_parser.esm_parser.**determine_computer_from_hostname**()

Determines which yaml config file is needed for this computer

Notes

The supercomputer must be registered in the `all_machines.yaml` file in order to be found.

Returns A string for the path of the computer specific yaml file.

Return type str

`esm_parser.esm_parser.`**`determine_regex_list_match`**(*test_str*, *regex_list*)

`esm_parser.esm_parser.`**`determine_set_variables_in_choose_block`**(*config*, *valid_model_names*, *model_name=[]*)

Given a config, figures out which variables are resolved in a choose block.

In order to avoid cyclic dependencies, it is necessary to figure out which variables are set in which choose block. This function recurses over all key/value pairs of a configuration, and for any key which is a model name, it determines which variables are set in it's `choose_` blocks. Tuples of `(model_name, var_name)` are appended to a list, which is returned with all it's duplicates removed.

Parameters

- **`config`** (*`dict`*) –
- **`valid_model_names`** (*`list`*) –
- **`model_name`** (*`list`*) –

Returns **set_variables** – A list of tuples of model_name and corresponding variable that are determined in `config`

Return type list

`esm_parser.esm_parser.`**`dict_merge`**(*dct*, *merge_dct*)

Recursive dict merge. Inspired by :meth:`dict.update()`, instead of updating only top-level keys, dict_merge recurses down into dicts nested to an arbitrary depth, updating keys. The `merge_dct` is merged into `dct`. :param dct: dict onto which the merge is executed :param merge_dct: dct merged into dct :return: None

`esm_parser.esm_parser.`**`do_math_in_entry`**(*tree*, *rhs*, *config*)

`esm_parser.esm_parser.`**`find_add_entries_in_config`**(*mapping*, *model_name*)

`esm_parser.esm_parser.`**`find_key`**(*d_search*, *k_search*, *exc_strings=''*, *level=''*, *paths2finds=[]*, *sep='.'*)

Searches for a key inside a nested dictionary. It can search for an integer, or a piece of string. A list of strings can be given as an input to search for keys containing all of them. An additional list of strings can be specified for keys containing them be excluded from the findings. This is a recursive function.

Note: Always define paths2finds, to avoid expansion of this list with consecutive calls.

Parameters

- **`d_search`** (*`dict`*) – The dictionary to be explored recursively.
- **`k_search`** (*`list, str, int`*) – String, integer or list of strings to be search for in `d_search`.
- **`exc_strings`** (*`list, str`*) – String or list of strings for keys containing them to be excluded from the finds. When set to an empty string, nothing is excluded.
- **`level`** (*`string`*) – String specifying the full path to the currently evaluated dictionary. Each dictionary level in these strings is separated by a `.`.

- **paths2finds** (*list*) – List of strings specifying the full path to the found keys in `d_search`. Each dictionary level in these strings is separated by a the specified string in `sep` (default is `"."`).
- **sep** (*string*) – String separator used in between each path component in `paths2finds`.

Returns **paths2finds** – List of strings specifying the full path to the found keys in `d_search`. Each dictionary level in these strings is separated by a `..`

Return type list

esm_parser.esm_parser.**find_one_independent_choose**(*all_set_variables*)

Given a dictionary of `all_set_variables`, which comes out of the function `determine_set_variables_in_choose_block`, gives a list of task/variable dependencies to resolve in order to figure out the variable.

Parameters **all_set_variables** (*dict*) –

Returns **task_list** – A list of tuples comprising `(model_name, var_name)` in order to resolve one `choose_` block. This list is built in such a way that the beginning of the list provides dependencies for later on in the list.

Return type list

esm_parser.esm_parser.**find_remove_entries_in_config**(*mapping*, *model_name*, *models=[]*)

esm_parser.esm_parser.**find_value_for_nested_key**(*mapping*, *key_of_interest*, *tree=[]*)

In a dict of dicts, find a value for a given key

Parameters

- **mapping** (*dict*) – The nested dictionary to search through
- **key_of_interest** (*str*) – The key to search for.
- **tree** (*list*) – Where to start searching

Returns The value of key anywhere in the nested dict.

Return type value

Note: Behaviour of what happens when a key appears twice anywhere on different levels of the nested dict is unclear. The uppermost one is taken, but if the key appears in more than one item, I'd guess something ambigous occus...

---

esm_parser.esm_parser.**find_variable**(*tree*, *rhs*, *full_config*, *white_or_black_list*, *isblacklist*)

esm_parser.esm_parser.**finish_priority_merge**(*config*)

esm_parser.esm_parser.**initialize_from_shell_script**(*filepath*)

esm_parser.esm_parser.**initialize_from_yaml**(*filepath*)

esm_parser.esm_parser.**list_all_keys_starting_with_choose**(*mapping*, *model_name*, *ignore_list*, *isblacklist*)

Given a `mapping` (e.g. a `dict`-type object), list all keys that start with `"choose_"` on any level of the nested dictionary.

Parameters

- **mapping** (*dict*) – The dictionary to search through for keys starting with `"choose_"`

- **model_name** (*str*) –
- **ignore_list** (*list*) –

Returns **all_chooses** – A list of tuples for .... A dictionary containing all key, value pairs starting with "choose_".

Return type list

esm_parser.esm_parser.**list_all_keys_with_priority_marker**(*config*)

esm_parser.esm_parser.**list_to_multikey**(*tree*, *rhs*, *config_to_search*, *ignore_list*, *isblacklist*)
A recursive_run_function conforming func which puts any list based key to a multikey elsewhere. Sorry, that sounds confusing even to me, and I wrote the function.

Parameters

- **tree** (*list*) –
- **rhs** (*str*) –
- **config_to_search** (*dict*) –

Notes

Internal variable definitions in this function; based upon the example: prefix_[[streams–>STREAM]]_postfix

- ok_part: prefix_
- actual_list: streams-->STREAM
- key_in_list: streams
- value_in_list: STREAM
- entries_of_key: list of actual chapter streams, e.g. [accw, echam6, e6hrsp, ...]

esm_parser.esm_parser.**look_for_file**(*model*, *item*)

esm_parser.esm_parser.**mark_dates**(*tree*, *rhs*, *config*)
Adds the DATE_MARKER to any entry who's key ends with "date"

esm_parser.esm_parser.**marked_date_to_date_object**(*tree*, *rhs*, *config*)
Transforms a marked date string into a Date object

esm_parser.esm_parser.**merge_dicts**(**dict_args*)
Given any number of dicts, shallow copy and merge into a new dict, precedence goes to key value pairs in latter dicts.

Note that this function only merges the first level. For deeper merging, use priority_merge_dicts.

Parameters ***dict_args** – Any number of dictionaries to merge together

Returns

Return type A merged dictionary (shallow)

esm_parser.esm_parser.**new_deep_update**(*receiving_dict*, *dict_to_be_included*, *winner='receiving'*, *blackdict={}*)

esm_parser.esm_parser.**new_dict_merge**(*dct*, *merge_dct*, *winner='to_be_included'*)
Recursive dict merge. Inspired by :meth:dict.update(), instead of updating only top-level keys, dict_merge recurses down into dicts nested to an arbitrary depth, updating keys. The merge_dct is merged into dct. :param dct: dict onto which the merge is executed :param merge_dct: dct merged into dct :param winner: should be either receiving (default) or to_be_included :return: None

esm_parser.esm_parser.**perform_actions**(*tree*, *rhs*, *config*)

esm_parser.esm_parser.**pprint_config**(*config*)
Prints the dictionary given to the stdout in a nicely formatted YAML style.

Parameters **config** (*dict*) – The configuration to print

Returns

Return type None

esm_parser.esm_parser.**priority_merge_dicts**(*first_config*, *second_config*, *priority='first'*)
Given two dictionaries, merge them together preserving either first or last entries.

Parameters

- **first_config** (*dict*) –
- **second_config** (*dict*) –
- **priority** (*str*) – One of "first" or "second". Specifies which dictionary should be given priority when merging.

Returns **merged** – A dictionary containing all keys, with duplicate entries reverting to the dictionary given in "priority". The merge occurs across all levels.

Return type dict

esm_parser.esm_parser.**purify_booleans**(*tree*, *rhs*, *config*)

esm_parser.esm_parser.**recursive_get**(*config_to_search*, *config_elements*)
Recusively gets entries in a nested dictionary in the form `outer_key.middle_key.inner_key = value`

Given a list of config elements in the form above (e.g. the result of splitting the string `"outer_key.middle_key.inner_key".split(".")` on the dot), the value "value" of the innermost nest is returned.

Parameters

- **config_to_search** (*dict*) – The dictionary to search through
- **config_elements** (*list*) – Each part of the next level of the dictionary to search, as a list.

Returns

Return type The value associated with the nested dictionary specified by `config_elements`.

Note: This is actually just a wrapper around the function `actually_recursive_get`, which is needed to pop off standalone model configurations.

---

esm_parser.esm_parser.**recursive_run_function**(*tree*, *right*, *level*, *func*, **args*, ***kwargs*)
Recursively runs func on all nested dicts.

Tree is a list starting at the top of the config dictionary, where it will be labeled "top"

Parameters

- **tree** (*list*) – Where in the dictionary you are
- **right** – The value of the last key in *tree*
- **level** (*str, one of "mappings", "atomic", "always"*) – When to perform func

- **func** (*callable*) – An function to perform on all levels where the type of `right` is in `level`. See the Notes for how this function's call signature should look.
- ***args** – Passed to func
- ****kwargs** – Passed to func

Returns

Return type right

Note: The `func` argument must be a callable (i.e. a function) and **must** have a call signature of the following form:

```
def func(tree, right, *args, **kwargs)
```

---

esm_parser.esm_parser.**remove_entries_from_chapter**(*config*, *remove_chapter*, *remove_entries*)

esm_parser.esm_parser.**remove_entries_from_chapter_in_config**(*model_config*, *valid_model_names*, *setup_config*, *valid_setup_names*)

esm_parser.esm_parser.**remove_entry_from_chapter**(*remove_chapter*, *remove_entries*, *model_to_remove_from*, *model_with_remove_statement*, *model_config*, *setup_config*)

Deletes the entries specified by the user using the `remove_<chapter>` command contained in the chapter, that can be either a list or a dictionary. After the removals the `remove_<chapter>` command is cleaned up from the config.

Parameters

- **remove_chapter** (*str*) – A string specifying the path inside the config to reach the chapter where the entries to be removed are. The string is composed by `remove_` followed by the path where each nested chapter is separated by a `..`
- **remove_entries** (*list*) – The list of entries to be remove from the chapter.
- **model_to_remove_from** (*str*) – Indicates the main chapter inside config where removes need to take place (i.e. `computer`, `general`, `<model>`, ...).
- **model_with_remove_statement** (*str*) – Indicates the main chapter where the remove command is defined.
- **model_config** (*dict*) – Component-specific general configuration.
- **setup_config** (*dict*) – Setup-specific general configuration.

esm_parser.esm_parser.**resolve_basic_choose**(*config*, *config_to_replace_in*, *choose_key*, *blackdict={}*)

esm_parser.esm_parser.**resolve_choose**(*model_with_choose*, *choose_key*, *config*)

esm_parser.esm_parser.**resolve_choose_with_var**(*var*, *config*, *user_config={}*, *model_config={}*, *setup_config={}*)

Searches for a `choose_` block inside a model configuration `config`, in which `var` is defined, and then resolves ONLY the `var` (the other variables in the `choose_` remain untouched). Needed, for example, for being able to use `include_models` from a `choose_` before the general choose-resolve takes place (i.e. include `xios` component from `oifs.yaml` using a `choose_`).

Parameters

- **var** (*str*) – Name of the variable to be searched inside `choose_` blocks.
- **config** (*dict*) – Model configuration to be changed if the `var` is resolved by the `choose_`.
- **user_config** (*dict*) – User configuration, used to search for the selected case of the `choose_`.
- **model_config** (*dict*) – Component configuration, used to search for the selected case of the `choose_`.
- **setup_config** (*dict*) – Setup configuration, used to search for the selected case of the `choose_`.

esm_parser.esm_parser.**resolve_remove_and_add**(*workdict*)

esm_parser.esm_parser.**shell_file_to_dict**(*filepath*)
Generates a ~`ConfigSetup` from an old shell script.

See also ~`ShellscriptToUserConfig`

Parameters **filepath** (*str*) – The file to load

Returns The parsed config.

Return type *ConfigSetup*

esm_parser.esm_parser.**to_boolean**(*value*)

esm_parser.esm_parser.**unmark_dates**(*tree*, *rhs*, *config*)
Removes the `DATE_MARKER` to any entry who's entry contains the `DATE_MARKER`.

esm_parser.esm_parser.**user_error**(*error_type*, *error_text*, *exit_code=1*)
User-friendly error using `sys.exit()` instead of an `Exception`.

Parameters

- **error_type** (*str*) – Error type used for the error heading.
- **text** (*str*) – Text clarifying the error.
- **exit_code** (*int*) – The exit code to send back to the parent process (default to 1)

esm_parser.esm_parser.**user_note**(*note_heading*, *note_text*)
Notify the user about something. In the future this should also write in the log.

Parameters

- **note_heading** (*str*) – Note type used for the heading.
- **text** (*str*) – Text clarifying the note.

17.6.3 esm_parser.shell_to_dict module

Backwards compatability for old runscripts

esm_parser.shell_to_dict.**ShellscriptToUserConfig**(*runscript_path*)
Generates a User Config from an old Shellscript

esm_parser.shell_to_dict.**mini_recursive_run_func**(*config*, *func*)

esm_parser.shell_to_dict.**purify_cases**(*config*)

esm_parser.shell_to_dict.**remap_old_new_keys**(*config*)

17.6.4 esm_parser.yaml_to_dict module

exception esm_parser.yaml_to_dict.**EsmConfigFileError**(*fpath*, *yaml_error*)
Bases: Exception

Exception for yaml file containing tabs or other syntax issues.

An exception used when yaml.load() throws a yaml.scanner.ScannerError. This error occurs mainly when there are tabs inside a yaml file or when the syntax is incorrect. If tabs are found, this exception returns a user-friendly message indicating where the tabs are located in the yaml file.

Parameters **fpath** (*str*) – Path to the yaml file

esm_parser.yaml_to_dict.**check_changes_duplicates**(*yamldict_all*, *fpath*)
Checks for duplicates and conflicting _changes and add_:

1. Finds variables containing _changes (but excluding add_) and checks if they are compatible with the same _changes inside the same file. If they are not compatible returns an error where the conflicting variable paths are specified. More than one _changes type in a file are allowed but they need to be part of the same _choose and not be accessible simultaneously in any situation.
2. Checks if there is any variable containing add_ in the main sections of a file and labels it as incompatible if the same variable is found inside a choose_ block. add_<variable>``s are compatible as long as they are inside ``choose_ blocks, but if you want to include something as a default, please just do it inside the <variable>.

> **Warning:** add_<variable>``s are not checked for incompatibility when they are included inside ``choose_ blocks. Merging of these add_<variable>``s is done using ``deep_update, meaning that the merge is arbitrary (i.e. if two choose_ blocks are modifying the same variable using add_, the final value would be decided arbitrarily). It is up to the developer/user to make good use of add_``s inside ``choose_ blocks.

Parameters

- **yamldict_all** (*dict*) – Dictionary read from the yaml file
- **fpath** (*str*) – Path to the yaml file

esm_parser.yaml_to_dict.**check_duplicates**(*src*)
Checks that there are no duplicates in a yaml file, and if there are returns an error stating which key is repeated and in which file the duplication occurs.

Parameters

- **src** (*object*) – Source file object
- **Exceptions** –
- **----------** –
- **ConstructorError** – If duplicated keys are found, returns an error

esm_parser.yaml_to_dict.**find_last_choose**(*var_path*)
Locates the last choose_ on a string containing the path to a variable separated by ",", and returns the path to the choose_ (also separated by ",") and the case that follows the choose_.

Parameters **var_path** (*str*) – String containing the path to the last choose_ separated by ",".

Returns

- **path2choose** (*str*) – Path to the last choose_.

- **case** (*str*) – Case after the choose.

esm_parser.yaml_to_dict.**yaml_file_to_dict**(*filepath*)

Given a yaml file, returns a corresponding dictionary.

If you do not give an extension, tries again after appending one. It raises an EsmConfigFileError exception if yaml files contain tabs.

Parameters **filepath** (*str*) – Where to get the YAML file from

Returns A dictionary representation of the yaml file.

Return type dict

Raises

- ***EsmConfigFileError*** – Raised when YAML file contains tabs or other syntax issues.
- **FileNotFoundError** – Raised when the YAML file cannot be found and all extensions have been tried.

17.7 esm_profile package

Top-level package for ESM Profile.

17.7.1 Submodules

17.7.2 esm_profile.esm_profile module

esm_profile.esm_profile.**timing**(*f*)

17.8 esm_rcfile package

Top-level package for ESM RCFile.

17.8.1 Submodules

17.8.2 esm_rcfile.esm_rcfile module

Usage

This package contains functions to set, get, and use entries stored in the esmtoolsrc file.

To use ESM RCFile in a project:

```
import esm_rcfile
```

You can set specific values in the `~/.esmtoolsrc` with:

```
set_rc_entry(key, value)
```

For example:

```
>>> set_rc_entry("SCOPE_CONFIG", "/pf/a/a270077/Code/scope/configs/")
```

Retriving an entry:

```
>>> fpath = get_rc_entry("FUNCTION_PATH")
>>> print(fpath)
/pf/a/a270077/Code/esm_tools/esm_tools/configs
```

With a default value for a non-existing key:

```
>>> scope_config = get_rc_entry("SCOPE_CONFIG", "/dev/null")
>>> print(scope_config)
/dev/null
```

Without a default value, you get `EsmRcfileError`:

```
>>> echam_namelist = get_rc_entry("ECHAM_NMLDIR")
EsmRcFileError: No value for ECHAM_NMLDIR found in esmtoolsrc file!!
```

This error is also raised if there is no `~/.esmtoolsrc` file, and no default is provided.

You can also get the entire rcfile as a dict:

```
>>> rcdict = import_rc_file()
```

API Documentation

exception esm_rcfile.esm_rcfile.**EsmRcfileError**
Bases: Exception

esm_rcfile.esm_rcfile.**get_rc_entry**(*key*, *default=None*)
Gets a specific entry

Parameters

- **key** (*str*) –
- **default** (*str*) –

Returns Value for key, or default if provided

Return type str

Raises ***EsmRcfileError*** –

- Raised if key cannot be found in the rcfile and no default is provided * Raised if the esmtoolsrc file cannot be found and no default is provided.

esm_rcfile.esm_rcfile.**import_rc_file**()
Gets current values of the esmtoolsrc file

Returns A dictionary representation of the rcfile

Return type dict

esm_rcfile.esm_rcfile.**set_rc_entry**(*key*, *value*)
Sets values in esmtoolsrc

Parameters

- **key** (*str*) –

- **value** (*str*) –

Note: Using this functions modifies the `rcfile`; which is stored in the current user's home directory.

17.9 esm_runscripts package

Top-level package for ESM Runscripts.

17.9.1 Submodules

17.9.2 esm_runscripts.batch_system module

exception esm_runscripts.batch_system.**UnknownBatchSystemError**
Bases: Exception

Raise this exception when an unknown batch system is encountered

class esm_runscripts.batch_system.**batch_system**(*config*, *name*)
Bases: object

calc_requirements(*config*)

static calculate_requirements(*config*)

check_if_submitted()

static get_batch_header(*config*)

static get_environment(*config*)

static get_extra(*config*)

get_job_state(*jobid*)

get_jobid()

static get_run_commands(*config*)

static get_sad_filename(*config*)

static get_submit_command(*config*, *sadfilename*)

job_is_still_running(*jobid*)

static submit(*config*)

static write_simple_runscript(*config*)

17.9.3 esm_runscripts.cli module

A small wrapper that combines the shell interface and the Python interface

esm_runscripts.cli.**main**()

esm_runscripts.cli.**parse_shargs**()
The arg parser for interactive use

17.9.4 esm_runscripts.compute module

esm_runscripts.compute.**add_batch_hostfile**(*config*)

esm_runscripts.compute.**all_files_to_copy_append**(*config*, *model*, *filetype*, *categ*, *file_source*, *file_interm*, *file_target*)

esm_runscripts.compute.**color_diff**(*diff*)

esm_runscripts.compute.**compile_model**(*config*)
Compiles the desired model before the run starts

esm_runscripts.compute.**copy_files_to_thisrun**(*config*)

esm_runscripts.compute.**copy_files_to_work**(*config*)

esm_runscripts.compute.**copy_tools_to_thisrun**(*config*)
Copies the tools, namelists and runscripts to the experiment directory, making sure that they don't overwrite previously existing files unless the `-U` flag is used. :param config: Dictionary containing the configuration information. :type config: dict

esm_runscripts.compute.**create_new_files**(*config*)

esm_runscripts.compute.**initialize_experiment_logfile**(*config*)
Initializes the log file for the entire experiment.

Creates a file `${BASE_DIR}/${EXPID}/log/${EXPID}_${setup_name}.log` to keep track of start/stop times, job id numbers, and so on. Use the function `write_to_log` to put information in this file afterwards.

The user can specify `experiment_log_file` under the `general` section of the configuration to override the default name. Timestamps for each message are given by the section `experiment_log_file_dateformat`, or defaults to `Tue Mar 17 09:36:38 2020`, i.e. `"%c"`. Please use `stftime` compatable formats, as described here: https://strftime.org

Parameters **dict** – The experiment configuration

Returns As per convention for the plug-in system; this gives back the entire config.

Return type dict

Attention: Calling this has some filesystem side effects. If the run number in the general configuration is set to 1, and a file exists for `general.exp_log_file`; this file is removed; and re-initialized.

esm_runscripts.compute.**modify_files**(*config*)

esm_runscripts.compute.**modify_namelists**(*config*)

esm_runscripts.compute.**prepare_coupler_files**(*config*)

esm_runscripts.compute.**run_job**(*config*)

esm_runscripts.compute.**update_runscript**(*fromdir*, *scriptsdir*, *tfile*, *gconfig*, *file_type*)
Updates the script tfile in the directory scriptdir with the file in the directory fromdir if the update flag (-U) is used during the call of esm_runscripts. If that flag is not used and the source and target are different then raises a user-friendly error recommending to use the -U flag with the warning that the files will be overwritten. :param cls: Compute object. :type cls: obj :param fromdir: Path of the source. :type fromdir: str :param scriptsdir: Path of the target. :type scriptsdir: str :param tfile: Name of the script to be updated. :type tfile: str :param gconfig: Dictionary containing the general information about the compute task. :type gconfig: dict :param file_type: String specifying the nature of the file, only necessary for printing information

and for the error description.

Parameters

- **Exceptions** –
- **----------** –
- **UserError** – If the target and source are different and the -U flag is not used when calling esm_runscripts, returns an error.

17.9.5 esm_runscripts.coupler module

class esm_runscripts.coupler.**coupler_class**(*full_config*, *name*)
Bases: object

add_couplings(*full_config*)

add_files(*full_config*)

finalize(*destination_dir*)

prepare(*full_config*, *destination_dir*)

prepare_restarts(*full_config*)

print_config_files()

tidy(*full_config*)

17.9.6 esm_runscripts.database module

class esm_runscripts.database.**experiment**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

archive_folder

cpuh

exp_folder

expid

gb

id

static nicer_output(*run*)

outcome

run_timestamp

runtime

setup_name

timestamp

static topline()

17.9.7 esm_runscripts.database_actions module

esm_runscripts.database_actions.**database_basic_entry**(*config*)

esm_runscripts.database_actions.**database_entry**(*config*)

esm_runscripts.database_actions.**database_entry_check**(*config*)

esm_runscripts.database_actions.**database_entry_crashed**(*config*)

esm_runscripts.database_actions.**database_entry_start**(*config*)

esm_runscripts.database_actions.**database_entry_success**(*config*)

17.9.8 esm_runscripts.filelists module

esm_runscripts.filelists.**assemble**(*config*)

esm_runscripts.filelists.**assemble_intermediate_files_and_finalize_targets**(*config*)

esm_runscripts.filelists.**check_for_unknown_files**(*config*)

esm_runscripts.filelists.**choose_needed_files**(*config*)

esm_runscripts.filelists.**complete_all_file_movements**(*config*)

esm_runscripts.filelists.**complete_one_file_movement**(*config*, *model*, *filetype*, *movement*, *movetype*)

esm_runscripts.filelists.**complete_restart_in**(*config*)

esm_runscripts.filelists.**complete_sources**(*config*)

esm_runscripts.filelists.**complete_targets**(*config*)

esm_runscripts.filelists.**copy_files**(*config*, *filetypes*, *source*, *target*)

esm_runscripts.filelists.**create_missing_file_movement_entries**(*config*)

esm_runscripts.filelists.**get_method**(*movement*)

esm_runscripts.filelists.**get_movement**(*config*, *model*, *filetype*, *source*, *target*)

esm_runscripts.filelists.**globbing**(*config*)

esm_runscripts.filelists.**log_used_files**(*config*)

esm_runscripts.filelists.**rename_sources_to_targets**(*config*)

esm_runscripts.filelists.**replace_year_placeholder**(*config*)

esm_runscripts.filelists.**report_missing_files**(*config*)

esm_runscripts.filelists.**resolve_symlinks**(*file_source*)

esm_runscripts.filelists.**reuse_sources**(*config*)

esm_runscripts.filelists.**target_subfolders**(*config*)

17.9.9 esm_runscripts.helpers module

class esm_runscripts.helpers.**SmartSink**
Bases: object

A class for smart sinks that allow for logging (using logger from loguru), even if the file path of the log file is not yet defined. The actual sink is not the instanced object itself, but the method sink of the instance. The log record is saved in self.log_record and the log file is written using the path specified in self.path. If the path is not specified, the log is stored only in the self.log_record. When the path is finally specified, self.log_record is dumped into the log file and from that moment, any time logger logs something it will also be written into the file. To specify the path the method def_path needs to be used.

def_path(*path*)
Method to define the path of the file. Once the path is defined, the log record is written into the file.

Parameters **path** (*str*) – Path of the logging file.

sink(*message*)
The actual sink for loguru's logger. Once you define a logger level a sink needs to be provided. Standard sinks include file paths, methods, etc. Providing this method as a sink (logger.add(<name_of_the_instance>.sink, level="<your_level>", ...)) enables the functionality of the SmartSink object.

Parameters **message** (*str*) – String containing the logging message.

write_log(*message*, *wmode*)
Method to write the logs into the disk.

Parameters

- **message** (*str, list*) – String containing the logging message or list containing more than one logging message, to be written in the file.
- **wmode** (*str*) – Writing mode to choose among "w" or "a".

esm_runscripts.helpers.**assemble_log_message**(*config*, *message*, *message_sep=None*, *timestampStr_from_Unix=False*)
Assembles message for log file. See doc for write_to_log

esm_runscripts.helpers.**end_it_all**(*config*)

esm_runscripts.helpers.**evaluate**(*config*, *job_type*, *recipe_name*)

esm_runscripts.helpers.**vprint**(*message*, *config*)

esm_runscripts.helpers.**write_to_log**(*config*, *message*, *message_sep=None*)
Puts a message into the experiment log file

Parameters

- **message** (*list*) – A list of the message elements; which is joined by either (highest to lowest): 1) the message_sep argument passed to the method, 2) The user's chosen seperator, as written in config["general"]["experiment_log_file_message_sep"], 3) An empty space " ".
- **message_sep** (*None*) – The hard-coded message seperator to use; which ignores user choices.

Note: The user can control two things regarding the logfile format:

1) The datestamp formatting, whjich is taken from the config section general.experiment_log_file_dateformat.

2) The message seperators; taken from `general.experiment_log_file_message_sep`. Note that if the programmer passes a `message_sep` argument; this one wins over the user choice.

17.9.10 esm_runscripts.inspect module

`esm_runscripts.inspect.`**`cat_file`**(*full_filepath*)

`esm_runscripts.inspect.`**`dir_size`**(*somepath*)

`esm_runscripts.inspect.`**`inspect_config`**(*config*)

`esm_runscripts.inspect.`**`inspect_file`**(*config*)

`esm_runscripts.inspect.`**`inspect_folder`**(*config*)

`esm_runscripts.inspect.`**`inspect_namelists`**(*config*)

`esm_runscripts.inspect.`**`inspect_overview`**(*config*)

`esm_runscripts.inspect.`**`inspect_size`**(*config*)

`esm_runscripts.inspect.`**`run_job`**(*config*)

17.9.11 esm_runscripts.last_minute module

`esm_runscripts.last_minute.`**`apply_last_minute_changes`**(*config*)

class `esm_runscripts.last_minute.`**`last_minute_changes`**(*config*)
Bases: `object`

`esm_runscripts.last_minute.`**`restore_protected_last_minute_changes`**(*config*)

17.9.12 esm_runscripts.methods module

`esm_runscripts.methods.`**`set_global_attr`**(*fname*, *attribute*, *value*)

17.9.13 esm_runscripts.namelists module

`esm-runscripts` Core Plugins for dealing with Fortran Namelists.

Provides plugins for loading, modifying, deleting, and writing Fortran Namelists as part of the `esm-runscripts` recipe. All plugins are found under the class Namelist as static methods. A deprecated class `namelist` (small "n") is provided, which warns you when it is used.

class `esm_runscripts.namelists.`**`Namelist`**
Bases: `object`

Methods for dealing with FORTRAN namelists

`static apply_echam_disturbance`(*config*)
Applies a disturbance to the DYNCTL chapter of the echam namelist via the enstdif

Relevant configuration entries: * disturbance_years (list of int): Which year to apply the disturbance * distrubance (float): Value to apply. Default can be found in echam.yaml

`static nmls_finalize`(*mconfig*, *verbose*)
Writes namelists to disk after all modifications have finished.

User Information

Part of the main log output will be a section specifing the actual namelists that have been used for your simulation, including all relevant additions, removals, or changes.

Programmer Information

A copy of the f90nml object representations of the namelists is stored under the dictionary key "namelist_objs", as a dictionary of ("namelist_name", f90nml_objfect) key/value pairs.

> **Warning:** Removing this step from your recipe might result in a broken run, as the namelists will not be present in their desired form! Even if your model runs, it might not contain all user-required changes.

Parameters **mconfig** (*dict*) – The model (e.g. ECHAM, FESOM, NEMO or OIFS) configuration

Returns **mconfig** – The modified configuration.

Return type dict

static nmls_load(*mconfig*)

Loads Fortran namelists into the configuration dictionary.

User Information

To associate namelists with a specific model, you should have a section in your configuration that lists the namelists:

```
fesom:
    namelists:
        - "namelist.config"
        - "namelist.oce"
        - "namelist.ice"
        - "namelist.diag"
```

Programmer Information

The namelists are represented by f90nml Namelist objects, and are stored under:

```
mconfig["namelists"]["namelist.echam"]``
```

This would point to the ECHAM namelist as a f90nml object, which closely resembles a dictionary.

The actual namelists to load are listed in the raw configuration as a list of strings:

```
mconfig['namelists'] = ['nml1', 'nml2', 'nml3', ...]
```

Namelists are assumed to have been copied to `mconfig["thisrun_config_dir"]`, and are loaded from there.

If the `mconfig` has a key `"namelist_case"` equal to "uppercase", the uppercase attribute of the f90nml representation of the namelist is set to `True`.

Parameters **mconfig** (*dict*) – The model (e.g. ECHAM, FESOM, NEMO or OIFS) configuration

Returns **mconfig** – The modified configuration.

Return type dict

static nmls_modify(*mconfig*)
Performs namelist changes.

User Information

In the configuration file, you should have a section as:

```
echam:
    namelist_changes:
        namelist.echam:
            radctl:
                co2vmr: 1200e-6
```

This would change the value of the echam namelist (namelist.echam), subsection radctl, entry co2vmr to the value 1200e-6.

Programmer Information

IDEA(PG): Maybe we can provide examples of how these functions are used in the code?

Note: Actual changes are performed by the f90nml package patch fuction. See here: https://tinyurl.com/y4ydz363

Parameters **mconfig** (*dict*) – The model (e.g. ECHAM, FESOM, NEMO or OIFS) configuration

Returns **mconfig** – The modified configuration.

Return type dict

static nmls_output(*mconfig*)

static nmls_output_all(*config*)

static nmls_remove(*mconfig*)
Removes an element from a namelist chapter.

User Information

In the configuration file, assume you have:

```
echam:
    namelist_changes:
        namelist.echam:
            radctl:
                co2vmr: "remove_from_namelist"
```

In this case, the entry co2vmr would be deleted from the radctl section of namelist.echam.

Programmer Information

IDEA(PG): Maybe we can provide examples of how these functions are used in the code?

Parameters **mconfig** (*dict*) – The model (e.g. ECHAM, FESOM, NEMO or OIFS) configuration

Returns **mconfig** – The modified configuration.

Return type dict

class esm_runscripts.namelists.**namelist**(**args*, ***kwargs*)

Bases: *esm_runscripts.namelists.Namelist*

Legacy class name. Please use Namelist instead!

17.9.14 esm_runscripts.oasis module

class esm_runscripts.oasis.**oasis**(*nb_of_couplings=1*, *coupled_execs=['echam', 'fesom']*, *runtime=1*, *debug_level=1*, *nnorest='F'*, *mct_version='4.0'*, *lucia=False*)

Bases: object

add_coupling(*lefts*, *lgrid*, *rights*, *rgrid*, *direction*, *transformation*, *restart_file*, *time_step*, *lresume*, *export_mode='DEFAULT'*)

add_input_coupling(*field_name*, *freq*, *field_filepath*)

add_output_file(*lefts*, *rights*, *leftmodel*, *rightmodel*, *config*)

add_restart_files(*restart_file*, *fconfig*)

finalize(*destination_dir*)

prepare_restarts(*restart_file*, *all_fields*, *model*, *config*)

print_config_files()

17.9.15 esm_runscripts.postprocess module

esm_runscripts.postprocess.**run_job**(*config*)

17.9.16 esm_runscripts.prepare module

esm_runscripts.prepare.**add_submission_info**(*config*)

esm_runscripts.prepare.**check_model_lresume**(*config*)

esm_runscripts.prepare.**finalize_config**(*config*)

esm_runscripts.prepare.**find_last_prepared_run**(*config*)

esm_runscripts.prepare.**initialize_batch_system**(*config*)

esm_runscripts.prepare.**initialize_coupler**(*config*)

esm_runscripts.prepare.**model_env_into_computer**(*config*)

This function allows to store in the computer dictionary, variables that were defined inside environment_changes or compile/runtime_environment_changes in the components.

It excludes module_actions and export_vars dictionaries as those are resolved later.

This function is necessary for controlling `choose_` blocks in the computer file from the component configuration file (i.e. add `useMPI` case to the component to control which `useMPI` case is selected in the computer file).

This function works both for compilation time and run time, and the result is that all components work under the same environment. The only exception is for the compilation of components, where `add_export_vars` and `add_module_actions` are excluded from the merging into `computer`, and are included individually in respective compilation scripts.

Later on, it might be desirable to always split the environments both for compiling (done by Paul Gierz, but this function would need to be adapted) and running (not done yet).

If this script gives you problems contact Miguel Andres-Martinez (miguel.andres-martinez@awi.de).

Parameters **config** (*dict*) – Dictionary containing the simulation/compilation information

Raises **User Note/Error** – If the same variable is found in two or more different component environments. Asks the user how to proceed.

esm_runscripts.prepare.**resolve_some_choose_blocks**(*config*)

esm_runscripts.prepare.**run_job**(*config*)

esm_runscripts.prepare.**set_leapyear**(*config*)

esm_runscripts.prepare.**set_logfile**(*config*)

esm_runscripts.prepare.**set_most_dates**(*config*)

esm_runscripts.prepare.**set_overall_calendar**(*config*)

esm_runscripts.prepare.**set_parent_info**(*config*)

esm_runscripts.prepare.**set_prev_date**(*config*)

esm_runscripts.prepare.**set_restart_chunk**(*config*)

17.9.17 esm_runscripts.sim_objects module

Documentation goes here

class esm_runscripts.sim_objects.**SimulationSetup**(*command_line_config=None*, *user_config=None*)

Bases: `object`

add_esm_runscripts_defaults_to_config(*config*)

compute(*kill_after_submit=True*)
All steps needed for a model computation.

Parameters **kill_after_submit** (*bool*) – Default `True`. If set, the entire Python instance is killed with a `sys.exit()` as the very last after job submission.

distribute_per_model_defaults(*config*)

get_total_config_from_user_config(*user_config*)

get_user_config_from_command_line(*command_line_config*)

inspect()

postprocess()
Calls post processing routines for this run.

tidy()
Performs steps for tidying up a simulation after a job has finished and submission of following jobs.

This method uses two lists, `all_files_to_copy` and `all_listed_filetypes` to sort finished data from the **current run folder** back to the **main experiment folder** and submit new **compute** and **post-process** jobs. Files for `log`, `mon`, `outdata`, and `restart_out` are gathered. The program waits until the job completes or an error is found (See ~self.wait_and_observe). Then, if necessary, the coupler cleans up it's files (unless it's a standalone run), and the files in the lists are copied from the **work folder** to the **current run folder**. A check for unknown files is performed (see ~self.check_for_unknown_files), files are moved from the the **current run folder** to the **main experiment folder**, and new compute and post process jobs are started.

Warning: The date is changed during this routine! Be careful where you put any calls that may depend on date information!

Note: This method is also responsible for calling the next compute job as well as the post processing job!

viz(*kill_after_submit=True*)
Starts the Viz job.

Parameters **kill_after_submit** (*bool*) – Default `True`. If set, the entire Python instance is killed with `sys.exit()`.

17.9.18 esm_runscripts.slurm module

Contains functions for dealing with SLURM-based batch systems

class esm_runscripts.slurm.**Slurm**(*config*)
Bases: `object`

Deals with SLURM, allowing you to check if a job is submitted, get the current job ID, generate a srun hostfile, get the current job state, and check if a job is still running.

filename
The filename for srun commands, defaults to `hostfile_srun`

Type str

path
Full path to this file, defaults to `thisrun_scripts_dir / filename`

Type str

Parameters **config** (*dict*) – The run configuration, needed to determine where the script directory for this particular run is.

calc_requirements(*config*)
Calculates requirements and writes them to `self.path`.

static check_if_submitted()
Determines if a job is submitted in the currently running shell by checking for `SLURM_JOB_ID` in the environment

Returns

Return type bool

static get_job_state(*jobid*)
Returns the jobstate full name. See `man squeue`, section `JOB STATE CODES` for more details.

Parameters **jobid** – `str` or `int`. The SLURM job id as displayed in, e.g. `squeue`

Returns The short job state.

Return type str

static get_jobid()
Gets the current SLURM JOB ID

Returns

Return type str or None

static job_is_still_running(*jobid*)
Returns a boolean if the job is still running

17.9.19 esm_runscripts.tidy module

class esm_runscripts.tidy.**RunFolders**(*config*)
Bases: `list`

Logs the `run_` directories in `<experiment_id>/log/run_folders.log`, updating it with new folders. The resulting object is a list of `run_` paths that exist or existed during the run time (even if they got deleted). This is useful for indexing operations such as `<object_name>[::<interval>]` used when removing `run_` folders. .. rubric:: Notes

It keeps the folder names sorted so there is no need of sorting out of the object, and it also prevents the existence of duplicates.

load()
Loads the existing paths of the `run_` folders.

save()
Saves all folder names.

update()
Updates the folders read from the log file with the currently existing folders, removes duplicates, sorts them and save them into the log file.

esm_runscripts.tidy.**all_done**(*config*)

esm_runscripts.tidy.**assemble_error_list**(*config*)

esm_runscripts.tidy.**check_for_errors**(*config*)

esm_runscripts.tidy.**clean_run_dir**(*config*)
This plugin allows you to clean up the `run_${DATE}` folders. To do that you can use the following variables under the `general` section of your runscript (documentation follows order of code as it is executed):

- `clean_runs`: **This is the most important variable for most users**. It can take the following values:
  - `True`: removes the `run_` directory after each run (**overrides every other** `clean_` **option**).
  - `False`: does not remove any `run_` directory (default) if no `clean_` variable is defined.
  - `<int>`: giving an integer as a value results in deleting the `run_` folders except for the last <int> runs (recommended option as it allows for debugging of crashed simulations).

Note: `clean_runs: (bool)` is incompatible with `clean_this_rundir` and `clean_runs: (int)` is incompatible with `clean_old_rundirs_except` (an error will be raised after the end of

the first simulation). The functionality of `clean_runs` variable **alone will suffice most of the standard user requirements**. If finer tunning for the removal of `run_` directories is required you can used the following variables instead of `clean_runs`.

- `clean_this_rundir`: (bool) Removes the entire run directory (equivalent to `clean_runs: (bool)`). `clean_this_rundir: True` **overrides every other** `clean_` **option**.
- `clean_old_rundirs_except`: (int) Removes the entire run directory except for the last <x> runs (equivalent to `clean_runs: (int)`).
- `clean_old_rundirs_keep_every`: (int) Removes the entire run directory except every <x>th run. Compatible with `clean_old_rundirs_except` or `clean_runs: (int)`.
- `clean_<filetype>_dir`: (bool) Erases the run directory for a specific filetype. Compatible with all the other options.
- `clean_size`: (int or float) Erases all files with size greater than `clean_size`, must be specified in bytes! Compatible with all the other options.

Example

To delete all the `run_` directories in your experiment include this into your runscript:

```
general:
        clean_runs: True
```

To keep the last 2 `run_` directories:

```
general:
        clean_runs: 2
```

To keep the last 2 runs and every 5 runs:

```
general:
        clean_old_rundirs_except: 2
        clean_old_rundirs_keep_every: 5
```

esm_runscripts.tidy.**copy_all_results_to_exp**(*config*)

esm_runscripts.tidy.**copy_stuff_back_from_work**(*config*)

esm_runscripts.tidy.**get_last_jobid**(*config*)

esm_runscripts.tidy.**init_monitor_file**(*config*)

esm_runscripts.tidy.**job_is_still_running**(*config*)

esm_runscripts.tidy.**maybe_resubmit**(*config*)

esm_runscripts.tidy.**rm_r**(*path*)
Python equivalent of rm -r

Parameters **path** (*str*) – Path or directory to remove

esm_runscripts.tidy.**run_job**(*config*)

esm_runscripts.tidy.**signal_tidy_completion**(*config*)

esm_runscripts.tidy.**size_bytes_to_human**(*num*, *suffix='B'*)

esm_runscripts.tidy.**size_human_to_bytes**(*s*, *suffix='B'*)

esm_runscripts.tidy.**start_post_job**(*config*)

esm_runscripts.tidy.**start_various_jobtypes_after_compute**(*config*)

esm_runscripts.tidy.**throw_away_some_infiles**(*config*)

esm_runscripts.tidy.**tidy_coupler**(*config*)

esm_runscripts.tidy.**wait_and_observe**(*config*)

esm_runscripts.tidy.**wake_up_call**(*config*)

17.9.20 esm_runscripts.virtual_env_builder module

esm_runscripts.virtual_env_builder.**find_package**(*pkg*)

esm_runscripts.virtual_env_builder.**get_base_prefix_compat**()
Get base/real prefix, or sys.prefix if there is none.

esm_runscripts.virtual_env_builder.**in_virtualenv**()

esm_runscripts.virtual_env_builder.**venv_bootstrap**(*config*)
Bootstraps your run into a virtual environment

17.9.21 esm_runscripts.yac module

class esm_runscripts.yac.**yac**(*full_config*, *nb_of_couplings=1*, *coupled_models=['echam', 'fesom']*, *grids=['atmo', 'feom']*, *runtime=1*)
Bases: object

Generates the configuration file for YAC coupler.

add_coupling(*field*, *transient_id*, *direction*, *config*)

add_output_file(*lefts*, *rights*, *leftmodel*, *rightmodel*, *config*)

add_restart_files(*restart_file*, *fconfig*)

finalize(*destination_dir*)

prepare_restarts(*restart_file*, *all_fields*, *model*, *config*)

print_config_files()

17.10 esm_version_checker package

esm_version_checker - Mini package to check versions of diverse esm_tools software

17.10.1 Submodules

17.10.2 esm_version_checker.cli module

Console script for esm_version_checker.

class esm_version_checker.cli.**GlobalVars**
Bases: object

A struct-like class for holding the global variables. GlobalVars instance should only be updated by the main function and should be 'read-only' by the other functions

from_github
top-level command-line option flag for connecting to the GitHub repo

Type bool

esm_tools_github_url
repository URL of the ESM-Tools

Type str

esm_tools_installed
each key is the specidif ESM-Tools package and value is bool

Type dict

esm_tools_github_url = 'https://github.com/esm-tools/'

esm_tools_installed = {}

from_github = False

esm_version_checker.cli.**check_importable_tools**()

esm_version_checker.cli.**dist_is_editable**(*dist*)
Is distribution an editable install?

esm_version_checker.cli.**editable_dist_location**(*dist*)
Determines where an editable dist is installed

esm_version_checker.cli.**get_esm_package_attributes**(*tool*)
Gets the attributes of the ESM-Tools package

Parameters **tool** (*str*) – name of the ESM-Tools package

Returns **attr_dict** – dictionary of attributes.

Return type dict

esm_version_checker.cli.**get_esm_packages**()
Gets the list of the installed ESM-Tools packages either locally or from the GitHub repository

Returns **esm_tools_modules** – list of strings where each item corresponds to a ESM-Tools package name

Return type list

esm_version_checker.cli.**global_options_decorator**(*func*)
decorator function for the global option

esm_version_checker.cli.**pip_install**(*package*)

esm_version_checker.cli.**pip_or_pull**(*tool*, *version=None*)

esm_version_checker.cli.**pip_uninstall**(*package*)

esm_version_checker.cli.**pip_upgrade**(*package*, *version=None*)

esm_version_checker.cli.**report_single_package**(*package*, *version*, *file_path*, *branch*, *describe*)

Nice output similar to the tree command in Linux

esm_version_checker.cli.**user_owns**(*binary*)

True or False if user owns binary

CHAPTER

EIGHTEEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

18.1 Types of Contributions

18.1.1 Report Bugs

Report bugs at https://github.com/esm-tools/esm_tools/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

18.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

18.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

18.1.4 Write Documentation

ESM Tools could always use more documentation, whether as part of the official ESM Tools docs, in docstrings, or even on the web in blog posts, articles, and such.

18.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/esm-tools/esm_tools/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

18.2 Get Started!

Ready to contribute? Here's how to set up *esm-tools* packages for local development (see *Python Packages* for a list of available packages). Note that the procedure of contributing to the *esm_tools* package (see *Contribution to esm_tools Package*) is different from the one to contribute to the other packages (*Contribution to Other Packages*).

18.2.1 Contribution to esm_tools Package

1. Fork the *esm_tools* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone https://github.com/esm-tools/esm_tools.git
   ```

   (or whatever subproject you want to contribute to).
3. By default, `git clone` will give you the release branch of the project. You might want to consider checking out the development branch, which might not always be as stable, but usually more up-to-date than the release branch:

   ```
   $ git checkout develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.
5. When you're done making changes, check that your changes pass flake8:

   ```
   $ flake8 esm_tools
   ```

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

18.2.2 Contribution to Other Packages

1. Follow steps 1-4 in *Contribution to esm_tools Package* for the desired package, cloning your fork locally with:

```
$ git clone https://github.com/esm-tools/<PACKAGE>.git
```

2. Proceed to do a development install of the package in the package's folder:

```
$ cd <package's_folder>
$ pip install -e .
```

3. From now on when binaries are called, they will refer to the source code you are working on, located in your local package's folder. For example, if you are editing the package *esm_master* located in `~/esm_master` and you run `$ esm_master install-fesom-2.0` you'll be using the edited files in `~/esm_master` to install FESOM 2.0.

4. Follow steps 5-7 in *Contribution to esm_tools Package*.

18.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/dbarbi/esm_tools/pull_requests and make sure that the tests pass for all supported Python versions.

18.4 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

CHAPTER

NINETEEN

CREDITS

19.1 Development Lead

- Dirk Barbi <dirk.barbi@awi.de>
- Paul Gierz <paul.gierz@awi.de>
- Nadine Wieters <nadine.wieters@awi.de>
- Miguel Andrés-Martínez <miguel.andres-martinez@awi.de>
- Deniz Ural <deniz.ural@awi.de>

19.2 Project Management

- Luisa Cristini <luisa.cristini@awi.de>

19.3 Contributors

- Sara Khosravi <sara.khosravi@awi.de>
- Fatemeh Chegini <fatemeh.chegini@mpimet.mpg.de>
- Joakim Kjellsson <jkjellsson@geomar.de>
- Sebastian Wahl <swahl@geomar.de>
- ...

19.4 Beta Testers

- Tido Semmler <tido.semmler@awi.de>
- Christopher Danek <christopher.danek@awi.de>
- ...

CHAPTER

TWENTY

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y