ESM Tools r6.9 UserManual

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

Jul 01, 2022

CONTENTS:

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.

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`.

5. Run `esm_master` once. You should see a long list of available targets if everything works.

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:

```
$ cd /some/folder/you/wish/to/work/in
$ 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.

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 <PATH TO ESM TOOLS>/esm_tools/runscripts/fesom2
$ (your_favourite_editor) fesom2-ollie-initial-monthly.yaml
```

Notice that the examples exist with the endings `.yaml`.

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 `.run` 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_xxxxxxxxxxxx/work`, as well as the complete configuration used to generate the simulation, located at `$BASE_DIR/$EXP_ID/run_xxxxxxxxxxxx/log`.

10. Run the experiment:

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

That should really be it. Good luck!

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 Installing in an encapuslated environment

Based on an alternative installation procedure, that provides an esm-tools installation employing direnv (https://direnv.net/), you can now install various encapsulated versions of esm-tools alongside each other. These different installations do not impact each others' configuration. Consequently, they can coexist in peaceful harmony. In the suggested alternative installation method all configurations will reside within the base folder of a specific esm-tools version that you install. There is no dependency on configurations outside the installation directory of a specific esm-tools version, mitigating the potential for side effects if another version of esm tools is installed in parallel. To install esm-tools as suggested here, just follow the procedure outlined below. The steps to create the installation involve preparation of direnv, including setting up an environment that encapsulates potentially version-specific settings, creating a dedicated directory to which a specific version of esm-tools will be installed, and installing the the esm-tools via pip. The commands to be executed are (note comments for further explanation):

```
$ curl -sfL https://direnv.net/install.sh | bash # install direnv if not yet done - this␣
↪enables encapsulation and parallel use of different esm-tools versions
$ mkdir esm_tools_v6.1.10 #adjust version number as appropriate
$ cd esm_tools_v6.1.10/
  #create .envrc (e.g. via direnv edit .) and add information matching the result of the␣
↪cat command below
$ cat .envrc
  module load python3
  layout python
  module load gcc
$ pip install -U pip wheel
$ pip install esm-tools
```

3.3 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.

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
```

levante.dkrz.de:

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

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
```

aleph:

```
$ module load git
$ module load python
```

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 unload netcdf_c
$ module load anaconda3
```

4.2 Installing

1. 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
```

2. Inside the same login or profile file, add also the `module` commands necessary for the HPC system you are using (find the lines in the section above).

3. You can choose to source now your login or profile file, so that the `module` and `export` commands are run (e.g. `$ source ~/.bash_profile`).

4. 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
```

5. 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.

4.3 Upgrade ESM-Tools

To upgrade all the *ESM-Tools* packages you can run:

```
$ esm_versions upgrade
```

This will only upgrade the packages that are not installed in editable mode. Those, installed in editable mode will need to be upgraded using `git`.

You can also choose to upgrade specific packages by adding the package name to the previous command, i.e. to upgrade `esm_master`:

```
$ esm_versions upgrade esm_parser
```

Note: If there are version conflicts reported back at this point with some of the Python modules (i.e. `pkg_resources.ContextualVersionConflict:  (<package name>)`), try reinstalling that package: `pip install <package> --upgrade --ignore-installed`.

4.4 Uninstall ESM-Tools

To uninstall your current installation make sure you have the most recent version of `pip` available for your system:

```
$ python3 -m pip install -U pip
```

Then, you can use use the following command to uninstall all *ESM-Tools* packages:

```
$ 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 -ri ~/.local/bin/esm*
$ rm -ri ~/.local/lib/python3.<version>/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`.

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`.

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:
            [ ... ]
        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 in the environment applied only during compilation.
runtime_environment_changes	Changes in 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.

For more information about `esm_environment` package, please check *ESM Environment*.

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

(continues on next page)

```
                                        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
    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"

        [ ... ]
```

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.

File movements

Inside the file dictionaries realm, it is possible to specify the type of movement you want to carry out (among `copy`, `link` and `move`), for a specific file or file type, and for a given direction. By default all files are `copied` in all directions.

The syntax for defining a file movement for a given file type is:

```
<model>:
    file_movements:
        <file_type>:
            <direction1>: <copy/link/move>
            <direction2>: <copy/link/move>
            [ ... ]
```

where the `file_type` in one among the *File Dictionary Types*, and the `direction` one of the following ones:

Movement file directions	
`init_to_exp`	Initial files to the corresponding general folder
`exp_to_run`	From general to the corresponding run folder
`run_to_work`	From run to the work folder on that run
`work_to_run`	From the work folder to the corresponding run folder
`all_directions`	Directions not specifically defined, use this movement

It is also possible to do the same for specific files instead of for all files inside a `file_type`. The syntax logic is the same:

```
<model>:
    file_movements:
        <file_key>:
            <direction1>: <copy/link/move>
            <direction2>: <copy/link/move>
            [ ... ]
```

where `file_key` is the key you used to identify your file inside the `<file_type>_files`, having to add to it `_in` or `_out` if the file is a restart, in order to specify in which direction to apply this.

Movements specific to files are still compatible with the `file_type` option, and only the moves specifically defined for files in the `file_movements` will differ from those defined using the `file_type`.

6.2.10 Accessing Variables from the Previous Run (`prev_run`)

It is possible to use the `prev_run` dictionary, in order to access values of variables from the previous run, in the current run. The idea behind this functionality is that variables from the previous run can be called from the yaml files with a very similar syntax to the one that would be used for the current run.

The syntax for that is as follows:

```
<your_var>: ${prev_run.<path>.<to>.<the>.<var>.<in>.<the>.<previous>.<run>}
```

For example, let's assume we want to access the *time_step* from the previous run of a *FESOM* simulation and store it in a variable called *prev_time_step*:

```
prev_time_step: ${prev_run.fesom.time_step}
```

Note: Only the single previous simulation loaded

Warning: Use this feature only when there is no other way of accessing the information needed. Note that, for example, dates of the previous run are already available in the current run, under variables such as `last_start_date`, `parent_start_date`, etc.

YAML FILE HIERARCHY

7.1 Hierarchy of YAML configuration files

The following graph illustrates the hierarchy of the different YAML configuration files.
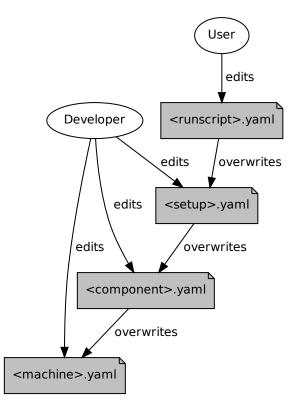
Fig. 1: ESM-Tools configuration files hierarchy

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.
heterogeneous_parallelization	A boolean that controls whether the simulation needs to be run with or without heterogeneous parallelization. When `false` OpenMP is not used for any component, independently of the value of `omp_num_threads` defined in the components. When `true`, `open_num_threads` needs to be specified for each component using OpenMP. `heterogeneous_parallelization` variable **needs to be defined inside the** `computer` section of the runscript. See *Heterogeneous Parallelization Run (MPI/OpenMP)* for examples.
omp_num_threads	A variable to control the number of OpenMP threads used by a component during an heterogeneous parallelization run. This variable **has to be defined inside the section of the components** for which OpenMP needs to be used. This variable will be ignored if `computer.heterogeneous_parallelization` is not set to `true`.

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/...>`).

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. Otherwise downloading ECHAM will already fail. To use the repository on gitlab.dkrz.de/modular_esm/echam.git, register for the MPI-ESM user forum at https://mpimet.mpg.de/en/science/modeling-with-icon/code-availability/mpi-esm-users-forum and send a screenshot to either dirk.barbi@awi.de, deniz.ural@awi.de or miguel.andres-martinez@awi.de

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 OpenIFS

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 OpenIFS. 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 TUX

Institute	wiki
Description	Tux image
Authors	who knows
Publications	`are you serious?`_
License	GPL

9.21 VILMA

9.22 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. . .

9.23 YAC

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

9.24 YAXT

Information	For more information about YAXT please . . .
Description	yaxt
Authors	
Publications	``_
License	

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`.

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`.

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 the specified run/step (i.e. to resume a second run you'll need to use `-r 2`).
-U, –update	[U]pdate the runscript in the experiment folder and associated files
–update-filetypes	Updates the requested files from external sources in a currently ongoing simulation. We strongly advise against using this option unless you really know what you are doing.
-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:
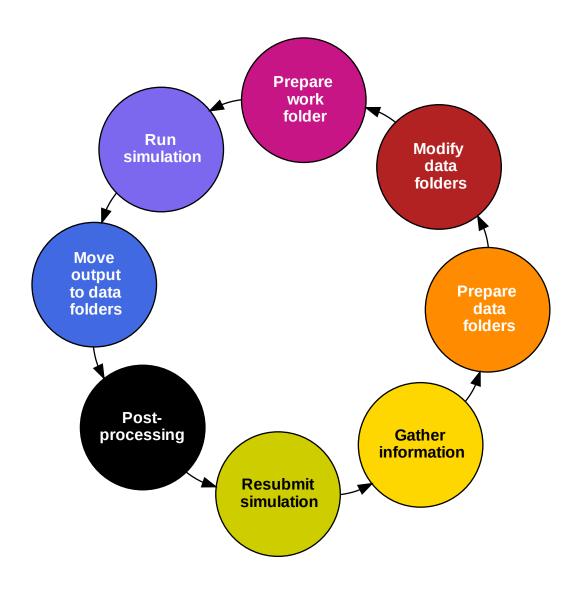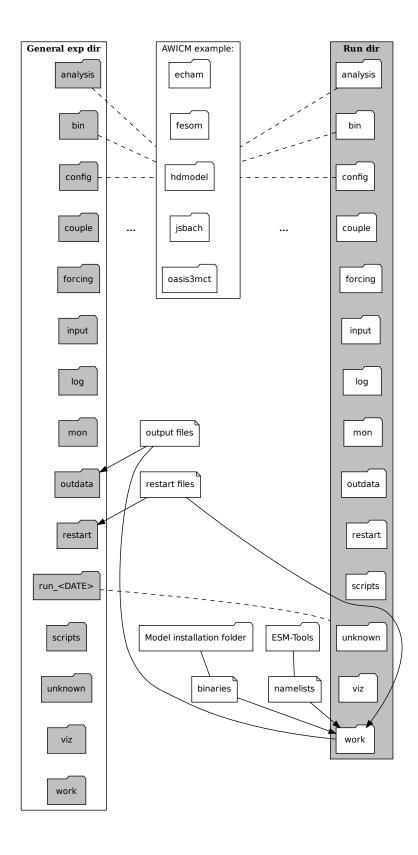
```
<general.base_dir>/<experiment_ID>
```

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>_ fin-ished_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.

ESM RUNSCRIPTS - USING THE WORKFLOW MANAGER

13.1 Introduction

Starting with Release 6.0, esm_runscripts allows the user to define additional subjobs for data processing, arrange them in clusters, and set the order of execution of these and the standard runjob parts in a flexible and short way from the runscript. This is applicable for both pre- and postprocessing, but especially useful for iterative coupling jobs, like e.g. coupling pism to vilma (see below). In this section we explain the basic concept, and the keywords that have to be set in the runscript to make use of this feature.

13.2 Subjobs of a `normal` run

Even before the addition of the workflow manager, the run jobs of esm_runscript were split into different subjobs, even though that was mostly hidden from the user's view. Before Release 6.0, these subjobs were:

```
compute --> tidy_and_resubmit (incl. wait_and_observe + resubmit next run)
```

Technically, `wait_and_observe` was part of the tidy_and_resubmit job, as was the resubmission, including above only for the purpose of demonstrating the difference to the new standard workflow, which is now (post-Release 6.0):

```
newrun --> prepcompute --> compute --> observe_compute --> tidy (+ resubmit next run)
```

Other than before adding the workflow manager, these standard subjobs are all separated and independant subjobs, each submitted (or started) by the previous subjob in one of three ways (see below). The splitting of the old compute job into newrun, prepcompute and compute on one side, and tidy_and_resubmit into observe and tidy, was necessary to enable the user to insert coupling subjobs for iterative coupling at the correct places. Here is what each of the standard subjobs does:

Subjob	Function
newrun	Initializes a new experiment, only very basic stuff, like creating (empty) folders needed by any of the following subjobs. NEEDS TO BE THE FIRST SUBJOB OF ANY EXPERIMENT.
prepcompute	Prepares the compute job. All the (Python) functionality that needs to be run, up to the job submission. Includes copying files, editing namelists, write batch scripts, etc.
compute	Actual model integration, nothing else. No Python codes involved.
observe_compute	Python job running at the same time as compute, checking if the compute job is still running, looking for some known errors for monitoring / job termination.
tidy	Sorts the produced outputs, restarts and log files into the correct folders, checks for missing and unknown files, builds coupler restart files if not present

It is important to understand that none of this has to be edited by the users, this is the default setup.

13.3 Keywords available for defining additional data processing sub-jobs

The workflow manager is intended to include shell scripted data processing jobs into the esm_runscripts workflow, so several things have to be defined:

- Name of the script to be run

- Name of the python script used for setting up the environment

- Name of the folder in which both of the above scripts can be found

- Information on how often the subjob should be called

- Information on between which other subjobs the new subjob should be inserted into the workflow

- In case it isn't clear: Which subjob should resubmit the next run.

The keywords used to define that are:

Key-word	Function
work-flow	Chapter headline in a model's section, indicating that alterations to the standard workflow will be defined here
sub-job_clusters	Section in the workflow chapter, containing the information on additional subjob_clusters. A subjob_cluster is a collection of subjobs run from the same batch script. Each subjob needs to belong to one cluster, if none is defined, each subjob will automatically get assigned to its own cluster. Each entry in `subjob_clusters` is a dict, with the outermost key being the (arbitrary) name of the cluster.
sub-jobs	Section in the workflow chapter, containing the information on additional subjobs.
run_after / run_before	Entry in spevifications of a subjob_cluster, to define before or after which other cluster of the workflow this cluster is supposed to run. Only one of the two should be specified. Can also be used in the specifications of subjobs if these subjobs get a corresponding cluster auto-assigned.
script:	
script_dir:	
call_function:	
env_preparation:	
next_run_triggered_by:	

13.4 Example 1: Adding an additional postprocessing subjob

In the case of a simple echam postprocessing job, the corresponding section in the runscript could look like this:

```
echam:
        [...other information...]

        workflow:
                next_run_triggered_by: tidy

                subjobs:
                        my_new_subjob:
                                nproc: 1
```

(continues on next page)

```
            run_after: tidy
            script_dir:
            script:
            call_function:
            env_preparation:
```

13.5 Example 2: Adding an additional preprocessing subjob

A preprocessing job basically is configured the same way as a postprocessing job, but the run_after entry is repl

13.6 Example 3: Adding a iterative coupling job

Writing a runscript for iterative coupling using the workflow manager requires some more changes. The principal idea is that each coupling step consists of two data processing jobs, one pre- and one postprocessing job. This is done this way as to make the coupling modular, and enable the modeller to easily replace one of the coupled components by a different implementation. This is of course up to the user to decide, but we generally advise to do so, and the iterative couplings distributed with ESM-Tools are organized this way.

ESM ENVIRONMENT

The package `esm_environment` takes care of generating the environments for the different HPCs supported by *ESM-Tools*. This is done through the use of the `EnvironmentInfos` class inside the different *ESM-Tools* packages.

For the correct definition of an environment for an HPC a *yaml* file for that system needs to be included inside the `esm_tools` package inside the `configs/machines/` folder (e.g. `ollie.yaml`). This file should contain all the required preset variables for that system and the environment variables `module_actions` and `export_vars`.

14.1 Environment variables

module_actions (list)
A list of `module` actions to be included in the compilation and run scripts generated by `esm_master` and `esm_runscripts` respectively, such as `module load netcdf`, `module unload netcdf`, `module purge`, etc. The syntax of this list is such as that of the command that would be normally used in shell, but omitting the `module` word, for example:

```
module_actions:
    - "purge"
    - "load netcdf"
```

This variable also allows for sourcing files by adding a member to the list such as `source <file_to_be_sourced>`.

export_vars (dict)
A dictionary containing all the variables (and their values) to be exported. The syntax is as follows:

```
export_vars:
    A_VAR_TO_BE_EXPORTED: the_value
```

The previous example will result in the following export in the script produced by `esm_master` or `esm_runscripts`:

```
export A_VAR_TO_BE_EXPORTED=the_value
```

As a dictionary, `export_vars` is not allowed to have repeated keys. This could be a problem when environments are required to redefine a variable at different points of the script. To overcome this limitation, repetitions of the same variable are allowed if the key is followed by an integer contained inside `[(int)]`:

```
export_vars:
    A_VAR_TO_BE_EXPORTED: the_value
    A_VAR_TO_BE_EXPORTED[(1)]: $A_VAR_TO_BE_EXPORTED:another_value
```

The resulting script will contain the following exports:

```
export A_VAR_TO_BE_EXPORTED=the_value
export A_VAR_TO_BE_EXPORTED=$A_VAR_TO_BE_EXPORTED:another_value
```

Note that the index is removed once the exports are transferred into the script.

14.2 Modification of the environment through the model/setup files

As previously mentioned, the default environment for a HPC system is defined inside its machine file (in `esm_tools/machines/<machine_name>.yaml`). However, it is possible to modify this environment through the model and/or coupled setup files (or even inside the runscript) to adjust to the model/setup requirements. For this purpose, the variables `environment_changes`, `compiletime_environment_changes` and `runtime_environment_changes` can be used.

environment_changes (dict)
> Allows for modifications of the machine `module_actions` and `export_vars`, both during compilation and runtime.

compiletime_environment_changes (dict)
> Allows for modifications of the machine `module_actions` and `export_vars`, only applied during compilation time.

compiletime_environment_changes (dict)
> Allows for modifications of the machine `module_actions` and `export_vars`, only applied during run time.

The syntax for this dictionary is the same as that defined in Environment variables, but using `add_` in front of the environment variables (`add_module_actions` and `add_export_vars`). Furthermore, the environment variables can be nested inside `choose_` blocks:

```
environment_changes:
        choose_computer:
                ollie:
                        add_export_vars:
                                COMPUTER_VAR: 'ollie'
                juwels:
                        add_export_vars:
                                COMPUTER_VAR: 'mistral'
```

Note: This changes are model-specific for compilation, meaning that **the changes will only occur for the compilation script of the model containing those changes**. For runtime, all the environments of the components will be added together into the same `.run` script. Please, refer to *Coupled setup environment control* for an explanation on how to control environments for a whole setup.

14.3 Coupled setup environment control

There are two ways in which the environments for the coupled setups can be modified: defining `environment_changes` for each component or defining a general `environment_changes` for the whole setup:

14.3.1 Component-by-component

The `environment_changes` are taken **from the standalone component files**. It is possible to modify these `environment_changes` through the setup file by including `environment_changes` **inside the chapter of that component**.

> **Warning:** Handling `environment_changes` in this fashion implies that compilation scripts can potentially end up containing different environments.

14.3.2 General environment for setups

To define a general `environment_changes` **for all the components of a setup**, include the `environment_changes` inside the `general` section of the setup file. **This will ignore all the** `environment_changes` **defined by the standalone files**. It is still possible to add component-specific `environment_changes` **from the component chapter inside the setup file**.

FIFTEEN

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 ;-)

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:

```
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"]`.

16.1 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.
sin- gle_proc_submit_flag	By default defined as `--ntasks-per-node=1`
addi- tional_flags	To add any additional flag that is not predefined in *ESM-Tools*

16.1.1 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:

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

16.2 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.

16.2.1 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
```

Fig. 1: disturb_years.dat location

16.2. Applying a temporary disturbance to ECHAM to overcome numeric instability (lookup table 69 overflows of various kinds)

16.2.2 See also

- ECHAM6 User Handbook, Table 2.4, dynctl

- Relevant source code

16.3 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>`

16.3.1 Example

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

Snippet

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

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

Full Runscript

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"
```

(continues on next page)

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

16.3.2 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*).

16.3.3 See also

- [Default namelists on GitHub](#)
- *Changing Namelists*
- *What Is YAML?*

16.4 Heterogeneous Parallelization Run (MPI/OpenMP)

Feature available since version: 5.1

In order to run a simulation with hybrid MPI/OpenMP parallelization include the following in your runscript:

1. Add `heterogenous_parallelization:   true` in the `computer` section of your runscript. If the `computer` section does not exist create one.

2. Add `omp_num_threads:   <number>` to the sections of the components you'd like to have OpenMP parallelization.

16.4.1 Example

AWICM3

In *AWICM3* we have 3 components: *FESOM-2*, *OpenIFS* and *RNFMAP*. We want to run *OpenIFS* with 8 OpenMP threads, *RNFMAP* with 48, and *FESOM-2* with 1. Then, the following lines need to be added to our runscript:

```yaml
general:
    [ ... ]
computer:
    heterogeneous_parallelization: true
    [ ... ]
fesom:
    omp_num_threads: 1
    [ ... ]
oifs:
    omp_num_threads: 8
    [ ... ]
rnfmap:
    omp_num_threads: 48
    [ ... ]
```

16.4.2 See also

- *Runtime variables*

16.5 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>
```

16.5.1 See also

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

16.6 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

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"
```

Multiple version files

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

```
[ ... ]

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':
```

(continues on next page)

```
      destination: fesom-1.4
      branch: co2_coupling
   '1.4-recom-mocsy-slp':
      branch: fesom-recom-mocsy-slp
      destination: fesom-1.4

[ ... ]
```

fesom-2.0.yaml

```
[ ... ]

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.

16.6.1 See also

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

16.7 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:
                            - "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*).

16.7.1 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

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
```

awicm.yaml

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

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



general:
        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
```

16.7.2 See also

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

16.8 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.

16.8.1 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
```

16.8.2 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>
   ```

16.8.3 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.

16.8.4 See also

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

16.9 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>
        - ...
```

16.9.1 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
```

16.9.2 See also

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

16.10 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.

16.10.1 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

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.

FESOM

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.

16.10.2 See also

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

16.11 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.

16.11.1 See also

- cookbook:How to run a branch-off experiment

FREQUENTLY ASKED QUESTIONS

17.1 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.

17.2 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.

17.3 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.:

```yaml
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

17.4 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]
```

(continues on next page)

```
UnicodeDecodeError: 'ascii' codec can't decode byte 0xf0 in position 1468: ordinal not␣
→in range(128)
        ----------------------------------------
 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*)

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:

18.1 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`).

18.2 esm_master.git

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

18.3 esm_runscripts.git

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

18.4 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.

18.5 esm_calendar.git

ESM TOOLS CODE DOCUMENTATION

19.1 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",
```

(continues on next page)

```
        "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.

19.1.1 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: `Base`

>   **exp_ref**

> **expid_id**

> **id**

> **tarballs**

class esm_archiving.database.model.**ArchivedFile**(***kwargs*)

> Bases: Base

> **fname**

> **id**

> **on_disk**

> **on_tape**

> **tarball**

> **tarball_id**

class esm_archiving.database.model.**Experiments**(***kwargs*)

> Bases: Base

> **archive**

> **created_at**

> **expid**

> **id**

class esm_archiving.database.model.**Tarball**(***kwargs*)

> Bases: 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*)

19.1.2 Submodules

19.1.3 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.

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

19.1.4 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)
        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*)

19.1.5 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.

19.2 esm_calendar package

Top-level package for ESM Calendar.

19.2.1 Submodules

19.2.2 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',
'Nov', 'Dec']

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.**date_range**(*start_date*, *stop_date*, *frequency*)

---

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

19.3 esm_cleanup package

Cleanup tool for ESM-Tools simulations

19.3.1 Submodules

19.3.2 esm_cleanup.cli module

esm_cleanup.cli.**evaluate_arguments**()

> The arg parser for interactive use

esm_cleanup.cli.**main**()

esm_cleanup.cli.**main_loop**(*folder*)

19.3.3 esm_cleanup.esm_cleanup module

esm_cleanup.esm_cleanup.**add_size_information**(*toplevel*, *item*)

esm_cleanup.esm_cleanup.**ask_for_action**(*folder*)

esm_cleanup.esm_cleanup.**assert_question**(*question*)

esm_cleanup.esm_cleanup.**check_if_folder_exists**(*folder*)

esm_cleanup.esm_cleanup.**dir_size**(*somepath*)

esm_cleanup.esm_cleanup.**file_size**(*somepath*)

esm_cleanup.esm_cleanup.**format_size**(*total_size*)

esm_cleanup.esm_cleanup.**inspect_size**(*thisfile*)

esm_cleanup.esm_cleanup.**is_experiment_folder**(*checkpath*)

esm_cleanup.esm_cleanup.**pick_experiment_folder**(*folder*)

esm_cleanup.esm_cleanup.**pick_subfolder**(*folder*)

esm_cleanup.esm_cleanup.**print_disclaimer**()

esm_cleanup.esm_cleanup.**print_folder_content**(*checkpath*, *saved_space*)

esm_cleanup.esm_cleanup.**read_in_yaml_file**(*filename*)

esm_cleanup.esm_cleanup.**remove_post_subfolders**(*folder*, *saved_space*)

esm_cleanup.esm_cleanup.**remove_run_subfolders**(*folder*, *saved_space*)

esm_cleanup.esm_cleanup.**remove_size_information**(*name_with_file*)

esm_cleanup.esm_cleanup.**remove_some_files**(*folder*, *saved_space*)

esm_cleanup.esm_cleanup.**remove_subfolder**(*folder*, *saved_space*)

19.4 esm_database package

Top-level package for ESM Database.

19.4.1 Submodules

19.4.2 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

19.4.3 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()**

19.4.4 esm_database.getch module

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

19.4.5 esm_database.location_database module

class esm_database.location_database.**database_location**(***kwargs*)

> Bases: Base
>
> **class_in**
>
> **id**
>
> **location**
>
> **table_name**
>
> **static topline()**

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

19.5 esm_environment package

19.5.1 Submodules

19.5.2 esm_environment.esm_environment module

19.6 esm_master package

Top-level package for ESM Master.

19.6.1 Submodules

19.6.2 esm_master.cli module

19.6.3 esm_master.compile_info module

19.6.4 esm_master.database module

class esm_master.database.**installation**(***kwargs*)

> Bases: Base
>
> **action**

    folder

    id

    static **nicer_output**(*run*)

    setup_name

    timestamp

    static **topline**()

19.6.5 esm_master.database_actions module

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

19.6.6 esm_master.esm_master module

19.6.7 esm_master.general_stuff module

19.6.8 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()

19.6.9 esm_master.task module

19.7 esm_motd package

19.7.1 Submodules

19.7.2 esm_motd.esm_motd module

19.8 esm_parser package

19.8.1 Submodules

19.8.2 esm_parser.esm_parser module

19.8.3 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

> > **Raises**
> > > **ConstructorError** – If duplicated keys are found, returns an error

esm_parser.yaml_to_dict.**check_for_empty_components**(*yaml_load*, *fpath*)

esm_parser.yaml_to_dict.**create_env_loader**(*tag='!ENV'*, *loader=<class 'yaml.loader.SafeLoader'>*)

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.

19.9 esm_plugin_manager package

19.9.1 Submodules

19.9.2 esm_plugin_manager.cli module

19.9.3 esm_plugin_manager.esm_plugin_manager module

19.10 esm_profile package

Top-level package for ESM Profile.

19.10.1 Submodules

19.10.2 esm_profile.esm_profile module

esm_profile.esm_profile.**timing**(*f*)

19.11 esm_runscripts package

19.11.1 Submodules

19.11.2 esm_runscripts.assembler module

19.11.3 esm_runscripts.batch_system module

19.11.4 esm_runscripts.chunky_parts module

19.11.5 esm_runscripts.cli module

19.11.6 esm_runscripts.compute module

19.11.7 esm_runscripts.config_initialization module

19.11.8 esm_runscripts.coupler module

19.11.9 esm_runscripts.database module

19.11.10 esm_runscripts.database_actions module

19.11.11 esm_runscripts.dataprocess module

19.11.12 esm_runscripts.event_handlers module

19.11.13 esm_runscripts.filelists module

19.11.14 esm_runscripts.helpers module

19.11.15 esm_runscripts.inspect module

19.11.16 esm_runscripts.last_minute module

19.11.17 esm_runscripts.logfiles module

19.11.18 esm_runscripts.methods module

19.11.19 esm_runscripts.mpirun module

19.11.20 esm_runscripts.namelists module

19.11.21 esm_runscripts.oasis module

19.11.22 esm_runscripts.observe module

19.11.23 esm_runscripts.pbs module

19.11.24 esm_runscripts.postprocess module

19.11.25 esm_runscripts.prepare module

19.11.26 esm_runscripts.prepcompute module

module, please refer to the handbook for user documentation as well as API documentation for the various sub-modules of the project.

Accessing Configuration

To access a particular configuration, you can use:

```
>>> from esm_tools import read_config_file
>>> ollie_config = read_config_file("machines/ollie")
```

Important note here is that the configuration file **has not yet been parsed**, so it's just the dictionary representation of the YAML.

esm_tools.**EDITABLE_INSTALL = False**

> Shows if the installation is installed in editable mode or not.
>
> > **Type**
> > bool

esm_tools.**caller_wrapper**(*func*)

esm_tools.**copy_config_folder**(*dest_path*)

esm_tools.**copy_namelist_folder**(*dest_path*)

esm_tools.**copy_runscript_folder**(*dest_path*)

esm_tools.**get_config_as_str**(*config*)

esm_tools.**get_config_filepath**(*config=''*)

esm_tools.**get_namelist_filepath**(*namelist=''*)

esm_tools.**get_runscript_filepath**(*runscript=''*)

esm_tools.**list_config_dir**(*dirpath*)

esm_tools.**read_config_file**(*config*)

> Reads a configuration file, which should be seperated by "/". For example, "machines/ollie" will retrieve the (unparsed) configuration of the Ollie supercomputer.
>
> > **Parameters**
> > **config** (`str`) – Configuration to get, e.g. machines/ollie.yaml, or echam/echam. You may omit the ".yaml" ending if you want, it will be appended automatically if not already there.
> >
> > **Returns**
> > A dictionary representation of the config.
> >
> > **Return type**
> > dict

esm_tools.**read_namelist_file**(*nml*)

> Reads a namelist file from a path, seperated by "/". Similar to `read_config_file`
>
> > **Parameters**
> > **nml** (`str`) – The namelist to load
> >
> > **Returns**
> > A string of the namelist file

> **Return type**
>> str

19.13.2 Submodules

19.13.3 esm_tools.cli module

19.14 esm_utilities package

Top-level package for ESM Utilities.

19.14.1 Submodules

19.14.2 esm_utilities.cli module

Console script for esm_utilities.

19.14.3 esm_utilities.esm_utilities module

Main module.

19.14.4 esm_utilities.utils module

esm_utilities.utils.**logfile_stats**(*logfile_to_read*)

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:

20.1 Types of Contributions

20.1.1 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.

20.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

20.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

20.1.4 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.

20.1.5 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 :)

20.2 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*).

20.2.1 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.

20.2.2 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*.

Get Back to the Standard Distribution

Once finished with the contribution, you might want to get back to the standard non-editable mode version of the package in the `release` branch. To do that please follow these steps:

1. Uninstall all *ESM-Tools* packages (*Uninstall ESM-Tools*). This will not remove the folder where you installed the package in editable mode, just delete the links to that folder.

2. Navigate to the `esm_tools` folder and run the `./install.sh` script.

3. Check that your package is now installed in the folder ~/.local/lib/python3.<version>/site-packages/.

Note: If the package is still shows the path to the editable-mode folder, try running `pip install --use-feature=in-tree-build .` from `esm_tools`.

20.3 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.

20.4 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
```

CREDITS

21.1 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>

21.2 Project Management

- Luisa Cristini <luisa.cristini@awi.de>

21.3 Contributors

- Sara Khosravi <sara.khosravi@awi.de>
- Fatemeh Chegini <fatemeh.chegini@mpimet.mpg.de>
- Joakim Kjellsson <jkjellsson@geomar.de>
- Sebastian Wahl <swahl@geomar.de>
- …

21.4 Beta Testers

- Tido Semmler <tido.semmler@awi.de>
- Christopher Danek <christopher.danek@awi.de>
- …

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e