Developer's documentation for the IOW Earth System Model

Sven Karsten,* Hagen Radtke†
Leibniz Institute for Baltic Sea Research Warnemünde (IOW)

February 8, 2022

About this document

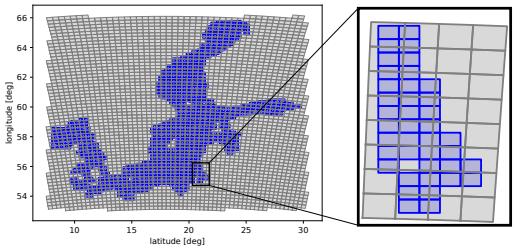
This document describes the details of the IOW Earth System Model. The Appendix sections contain more detailed information intended for developers. A guide for the initial steps to use the models is provided in the corresponding `Readme.md` file.

*sven.karsten@io-warnemuende.de
†hagen.radtke@io-warnemuende.de

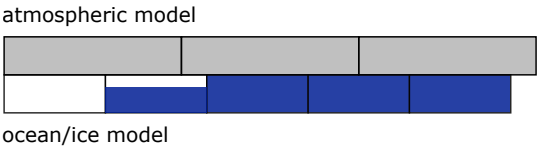
Table of Contents

1. Introduction and the coupling concept

One basic problem when dealing with regional climate models system is that the individual components (atmosphere, ocean, land etc.) are described by different models (realized as computer programs) that act on different grids, see Fig. 1a and Fig. 1b. Still, the components have to be coupled in order communicate their state to each other and exchange fluxes as given by nature itself.



(a) Overlaying grids of atmospheric and ocean model for the Baltic Sea.

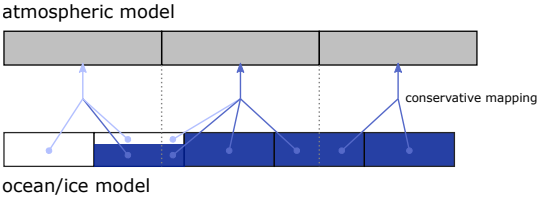


(b) Abstraction of figure 1a in one spatial horizontal dimension. Bottom model can support different surface types as water (blue) and ice (white).

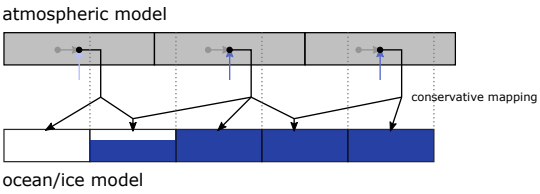
Figure 1: The different grids of the different models.

To exemplify this problem let us consider the calculation of a flux of something from the atmosphere to the bottom. Usually, if the flux depends on the state of the ocean, some state variables have to be communicated first to the atmosphere. Since the atmospheric grids are normally larger, the information has to be averaged (weighted by areas) over several bottom cells, over different surface types, see Fig. 2a.

With this averaged state information and its own internal state, the atmospheric model can now calculate the flux as a field on the atmospheric grid. It is noteworthy that the flux cannot be calculated for the different surface types differently, since the atmospheric model does not know about that (at least not without changing the code significantly). Finally, the flux field then has to be redistributed on the bottom cells (again in a area-weighted manner such that flux variable is overall conserved), see Fig. 2b. Since the flux is only calculated from averaged information and not surface-type-dependent, this approach locally not consistent and can become inaccurate. This is especially true if many bottom grid cells are covered by one atmospheric grid cell.



(a) Average of ocean's state variables communicated to the atmosphere.



(b) Calculation of fluxes in the atmospheric model and remapping on to the ocean's grid.

Figure 2: The standard way of coupling.

The alternative approach chosen within the IOW ESM is the introduction of a third component, i.e. the flux calculator that acts on the exchange grid. This grid is the set of intersections between the atmospheric and the bottom grid and thus has, by construction, a higher resolution than all involved model components, see Fig. 3.

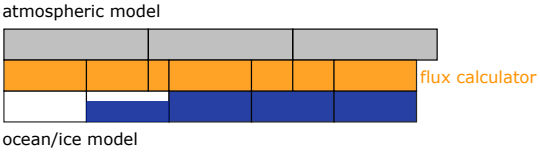
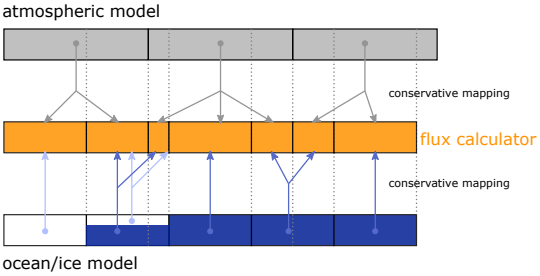


Figure 3: Introduction of the exchange grid on which the flux calculator is acting.

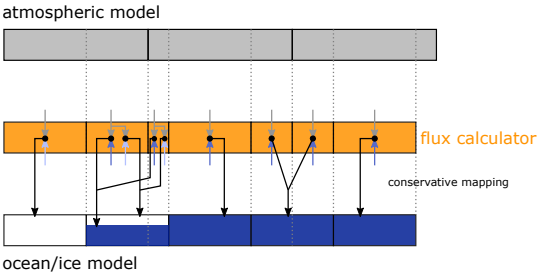
The example from above, i.e. a flux shall be communicated from the atmosphere to the ocean, is then treated as follows.

First, the physical components of the coupled model send their necessary state variables to the flux calculator. The variables are thereby mapped onto the exchange grid, see Fig. 4a. Importantly, since the exchange-grid cells are always smaller or equal to the „physical" grid cells, this mapping does not feature any averaging and, thus, no information is lost. Moreover, different surface types can be treated individually since the flux calculator might know about these feature of the bottom model.

Second, with all the state information, the flux calculator is now able to calculate the flux of interest. Any formula can be used that maps the given state variables onto the desired flux field. The calculation only requires local information and can be surface-type-dependent. The resulting flux has to be finally mapped onto the bottom grid, see Fig. 4b. Note that, although not shown in the figures, the very same fluxes are communicated to the atmospheric model as well. This ensures a conservative and consistent exchange of mass, energy and momentum between the different model components.



(a) Communication of state variables to the flux calculator. Importantly no averaging is performed while mapping.



(b) Fluxes are calculated and subsequently communicated to the bottom model.

Figure 4: Coupling the models via the exchange grid and the flux calculator.

2. Current implementation of the coupling concept

2.1. Components

The following components can be coupled in the IOW ESM:

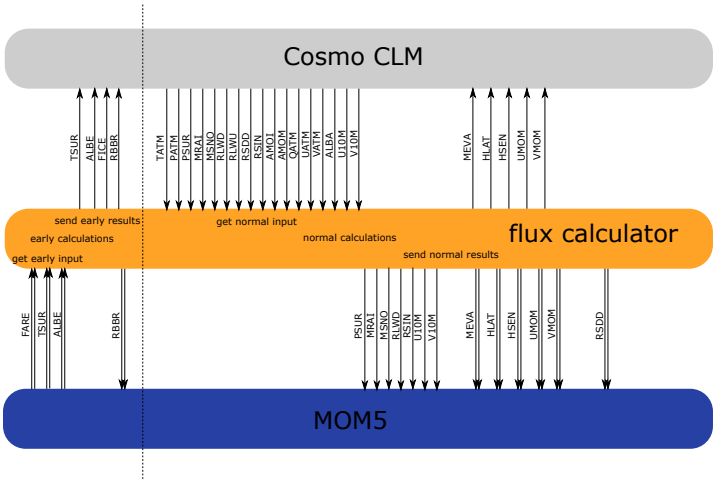
- COSMO-CLM atmospheric model, see here
- MOM5.1 ocean model (including (a) ERGOM ecosystem model and (b) dynamic ice model coupled internally via FMS coupler), see here

2.2. Coupling philosophy

The components are coupled via the OASIS3-MCT coupler (see here), but not directly:

1. The components (Atmosphere, Land, Ocean) communicate their variables (temperature, pressure, velocity etc.) to a flux calculator executable.
2. The flux calculator runs on an exchange grid and calculates fluxes (radiation, heat, mass, momentum).
3. The fluxes are back-communicated to the components (Atmosphere, Land, Ocean).

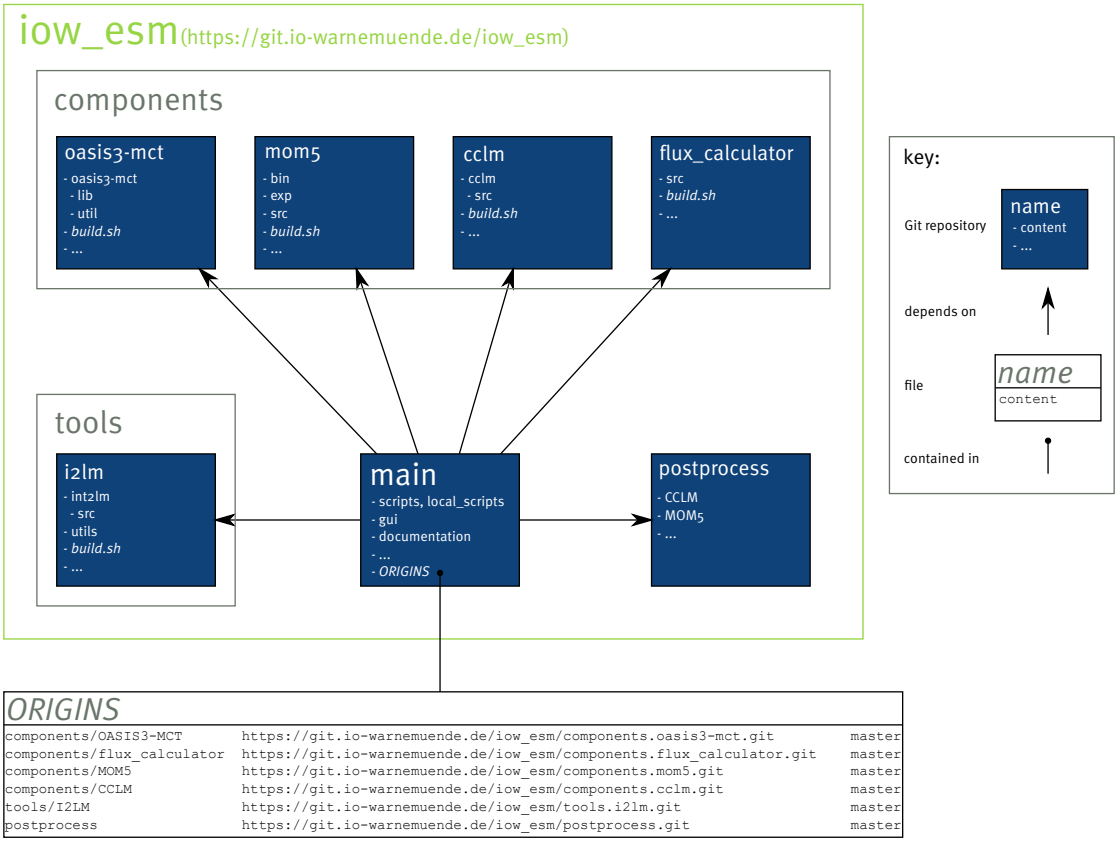
This procedure automatically ensures conservation of all exchanged quantities.



Some fluxes, however, do not fit into this concept. Precipitation and radiative fluxes will still be determined by the atmospheric model (on the coarse atmospheric grid) and the resulting fluxes will be sent to the flux calculator executable. Precipitation is then simply passed to the bottom models. Radiative fluxes need some redistribution between different surface areas under the same atmospheric grid cell (e.g. ocean and land) that show different albedo. This concept is described in detail in Section A.5.

Still, there is no direct communication the two physical components and this enables ultimately interchangeability of the models.

3. Design and structure of the IOW ESM project



ORIGINS

components/OASIS3-MCT	https://git.io-warnemuende.de/iow_esm/components.oasis3-mct.git	master
components/flux_calculator	https://git.io-warnemuende.de/iow_esm/components.flux_calculator.git	master
components/MOM5	https://git.io-warnemuende.de/iow_esm/components.mom5.git	master
components/CCLM	https://git.io-warnemuende.de/iow_esm/components.cclm.git	master
tools/I2LM	https://git.io-warnemuende.de/iow_esm/tools.i2lm.git	master
postprocess	https://git.io-warnemuende.de/iow_esm/postprocess.git	master

Figure 5: Structure of the project's source code management.

4. Getting and setting up the IOW ESM for first usage

How to get the sources of the IOW ESM and setting it up for a first run is described in the `Readme.md` file.

Once the steps described therein have been followed, the user finds a directory structure that is presented in Sec. 5.

5. Directory structure of the IOW ESM

Under a base directory which the user can select, there is the following directory structure:

```
components
   OASIS3-MCT
   CCLM
   MOM5
   flux_calculator
tools
   I2LM
input
   CCLM_MAINGRID
   MOM5_MAINGRID
   GETM_SUBGRID
scripts
   prepare
   run
postprocess
   MOM5
      task1
      task2
      ...
   CCLM
      task1
      task2
      ...
work
   CCLM_MAINGRID
   MOM5_MAINGRID
   GETM_SUBGRID
output
   RUN01
      CCLM_MAINGRID
         19500101
```

```
         19510101
         ...
      MOM5_MAINGRID
         ...
      GETM_SUBGRID
         ...
   RUN02
   ...
hotstart
   RUN01
      CCLM_MAINGRID
         19510101
         19520101
         ...
      MOM5_MAINGRID
         ...
      ...
   RUN02
   ...
```

5.1. The components directory

This directory contains the components of the IOW-ESM, which are

- COSMO-CLM, a regional atmospheric model
- MOM5, a hydrodynamic ocean model
- the OASIS3-MCT library for coupling the components via MPI
- the flux calculator, a separate executable to calculate fluxes between atmosphere and ocean/land on an exchange grid.

Each component's directory contains subdirectories for

- the source code,
- the compile scripts,
- the compiled libraries and/or executables.

Note that all components depend on the OASIS3-MCT library but not mutually on each other.

5.2. The tools directory

This directory contains components of the IOW-ESM that are not directly coupled to other components but provide useful input for the models. Currently the following tools are available

- int2lm, a part of the CCLM package to generate forcings from coarse data, see here

5.3. The input directory

The user of the model system has to create a directory called `input`. It contains subdirectories for the individual model instances which shall be coupled. For each model instance, the subdirectory shall be named `MODELNAME_GRIDNAME` where `MODELNAME` is the name of the model (as it is called in the `components` folder) and `GRIDNAME` is the name of the model grid, since more than one instance of a model can run at the same time but with different grids. Section 7.1 describes how to store model input in these folders.

In addition, there are three files directly in the `input` directory:

- `global_settings.py` contains settings for the entire model system, such as name and duration of the present run, coupling timestep, or whether the work directory in which the coupled model runs should be global or node-local.
- `flux_calculator.nml` is a configuration file for the `flux_calculator` executable which defines how fluxes between atmosphere and land/sea shall be calculated, i.e. which bulk formulae are used.
- `namcouple` is the OASIS3-MCT coupling configuration file and describes how data are exchanged between the model subcomponents. It needs to match the configuration in `flux_calculator.nml`. It can be thus generated automatically by setting the option `generate_namcouple = True` in `global_settings.py`. Importantly, for an uncoupled run you have to provide a `namcouple` file with `$NFIELDS` equal to zero.

5.4. The scripts directory

This directory contains scripts (bash or python3 scripts) which perform tasks in preparing model runs, running the model system, or

5.5. The postprocess directory

This directory contains scripts (bash and python) for postprocessing of data which is located in the output directory.

5.6. The work directory (automatically created during model start)

This is the directory in which the models are actually executed. It can also be located in another place, such as a RAM drive. If the model run fails, you can check for errors here.

5.7. The output directory (automatically created during model run)

For each run of the model system, there will be a separate directory which contains the model results. These are sorted by sub-model and by starting time.

5.8. The hotstart directory (automatically created during model run)

This contains hotstart files to restart the model from a specific date. The directory is sorted like the `output` folder.

6. Compiling IOW ESM components

It is important to compile the OASIS3-MCT libraries first before compiling the other subcomponents. This is even true if you want to build only the CCLM or the MOM5, respectively, for an uncoupled run. Whether a model runs coupled or uncoupled is only decided via the configuration files in the 'input' directory, see Sec. 7.1. The binaries are the same for both types of applications.

The correct order of building is ensured by using the `build.sh` script in the base directory. How this script is called correctly is written in detail in the `Readme.md` file.

However, it is still possible to build each component individually. In order to compile one of the components, go to the corresponding folder:

- `components/OASIS3-MCT`
- `components/CCLM`
- `components/MOM5`
- `components/flux_calculator`

This folder contains the source code and makefiles required to build this component. Execute the file `build.sh`, which will then create your executable. How this script is called correctly is written in detail in the `Readme.md` file. You will need to modify the file `build.sh` according to the machine you are compiling it on, and according to the compiler and MPI libraries which you want to use. For some machines that are typically used at the IOW the working build scripts are already provided in the repository. Which machines are supported and which steps you have to follow to register a new machine is written in detail in the `Readme.md` file.

For compiling in debug mode (with traceback), the build scripts can be called with an additional argument which will create a separate executable with debug options enabled. You can define which of the executables to use by specifying `debug_mode = True` or `False` in the file `global_settings.py`. You should obviously not do productions runs with the debug-mode executables since this will waste computational resources.

7. Preparing IOW ESM runs

7.1. Preparing the input folders for individual models

The IOW ESM supports to run a single atmospheric model and several ocean or land models at the same time. For each of these, a sub-folder in the `input` directory has to be prepared. It shall be named `MODELNAME_GRIDNAME` where `MODELNAME` is the name of the model (as it is called in the `components` folder) and `GRIDNAME` is the name of the model grid.

If you work on the machines typically used at the IOW, the `Readme.md` file explains in detail how and where to get working example setups.

7.1.1. General remarks

The input folders can contain files and arbitrarily named sub-directories which will simply be copied (or dynamically linked) to create a work directory for the model. An exception are subfolders named `from`. These should themselves contain subfolders named after years or dates as YYYYMMDD (e.g. `1950` or `19501201`). In case that the starting date of a model run is equal to or larger than a subfolder's date, the files inside this folder will be copied to the directory containing the folder `from`. For example, if there are files

```
input/MOM5_maingrid/OBC/from/1950/obc_sealevel.nc
input/MOM5_maingrid/OBC/from/1951/obc_sealevel.nc
input/MOM5_maingrid/OBC/from/1952/obc_sealevel.nc
```

and the starting date of a model run is 1951-06-01, the file
`input/MOM5_maingrid/OBC/from/1951/obc_sealevel.nc`
will be copied (or linked) to
`work/MOM5_maingrid/OBC/obc_sealevel.nc`.

7.1.2. Expected contents of the COSMO-CLM input directory

The following settings are required in `INPUT_PHY`:

```
llake = .FALSE.
lseaice = .FALSE.
```

For a coupled run the following settings are required in the `OASISCTL` block of `INPUT_OASIS`:

```
ytype_lsm = 'terra',
ytype_oce = 'flxcl',
CPL_FLG = 1,
dt_cp = *auto*
```

whereas for an uncoupled run it is mandatory to have

```
ytype_oce = 'nooce'
```

The following settings are required in `INPUT_DYN`:

```
l2tls = .TRUE.
```

7.1.3. Expected contents of a MOM5 input directory

For a coupled run the following settings in the `coupler_nml` block of `input.nml` are required

```
dt_cpld = *AUTO*
dt_atmos = *AUTO*
do_atmos = .false.
do_land = .false.
do_ice = .true.
do_ocean = .true.
type_atmos = 'flux_calculator'
```

whereas for an uncoupled run it is mandatory to have `type_atmos = 'none'`

7.2. Preparing the global settings

7.2.1. Modeller's information

```
#####################################################
# Global settings for the IOW-ESM model run         #
#####################################################

###################################
# STEP 1: Info about the modeller #
###################################
modeller        = "Sven Karsten"
email           = "sven.karsten@io-warnemuende.de"
institution     = "Leibniz Institute for Baltic Sea Research Warnemunde (IOW)"
```

7.2.2. Run information

```
###############################
# STEP 2: Info about the run  #
###############################
run_name        = "RUN14"        # name of the current run
run_description = "run for fixing land-sea mask issue"     # description: what
init_date       = "19800101"           # date when model is/was cold-started
final_date      = "20100101"           # when will the model run finally end?
                                       # choose final_date<init_date to make r
debug_mode      = False                # whether or not to use executables com
```

```
################################################
# STEP 3: Time stepping info                   #
################################################
coupling_time_step = 600                # time step when fluxes are calculated
run_duration = "1 month"                 # duration of one model run (day/days,
runs_per_job = 1                        # how many runs will be done in one job
max_attempts = 1                        # if a run fails, you can have new atte

################################################
# STEP 4: Working directory options            #
################################################
workdir_base = "work"                   # base directory where individual model
                                        # if it does not start with "/", it is
local_workdir_base = "${LOCAL_TMPDIR}"                # If you want the workdi
                                        # This can also be an environment varia
copy_to_global_workdir = True           # If local_workdir_base is given, use t
                                        # entire content of the local workdir w
                                        # Output and restarts will be collected
link_files_to_workdir = True            # TRUE: input files will be linked to t
                                        # FALSE: input files will be copied to

################################################
# STEP 5: Two-way coupling options             #
################################################
flux_calculator_mode = "single_core_per_bottom_model"   # "single_core_per_bot
#flux_calculator_mode = "none"                           # started in an MPI t
                                                        # "on_bottom_model_co
                                                        # same cores as the bo
                                                        # "none": no flux_calc

################################################
# STEP 6: Parallel execution options           #
################################################
mpi_run_command = 'mpirun -configfile mpmd_file'        # the shell command us
                                                        # it may contain the f
                                                        #   _NODES_ total numb
                                                        #   _CORES_ total numb
                                                        #   _THREADS_ total nu
                                                        #   _CORESPERNODE_ num
                                                        #   Examples: Intel MP
                                                        #             OpenMPI
mpi_n_flag = '-n'                                       # the mpirun flag for
                                                        #   Examples: Intel MP
                                                        #             OpenMPI
```

```
bash_get_rank = 'my_id=${PMI_RANK}'                          # a bash expression th
                                                             #   Examples: Intel MP
                                                             #             OpenMPI-
python_get_rank = 'my_id = int(os.environ["PMI_RANK"])'      # a python expression
                                                             #   Examples: Intel MP
                                                             #             OpenMPI-
cores_per_node = 40                                          # maximum number of co
jobscript_template = 'input/jobscript_hlrng'                 # path to a template f
                                                             # may contain the same
check_layout_only = True                                     # if set to True, will

resubmit_command = "source ~/.bash_profile; sbatch jobscript" # will be execut

################################################
# STEP 7 (optional): generate namcouple file automatically  #
################################################
generate_namcouple = True

################################################
# STEP 8 (optional): Start postprocessing after run has finished #
################################################
process_raw_output = True
```

7.3. Preparing a jobscript template

In order to start a run of the IOW ESM on a supercomputer, a jobscript template must be provided. If you have deployed your setup as described in the `Readme.md` file, you will have such a template in the `input` directory. This has to be adapted to your personal needs as follows.

1. Be sure that you leave the following lines unchanged
   `#SBATCH -nodes=_NODES_`
   `#SBATCH -ntasks=_CORES_`
   `#SBATCH -tasks-per-node _CORESPERNODE_`
   These will be filled later by the run scripts according to the parallelization layout.
2. Adjust the job name, the account, the compute partition and the runtime.
3. Go to the file `input/global_settings.py` and specify the filename of your template via the variable `jobscript_template`
4. Also in `input/global_settings.py`, fill in the number of cores per node to use.
5. Later you may want to check whether the parallelization is efficient, i.e. whether none of the components has too much idle time waiting for the others. This is described in Sec. 8.2
6. The actual jobscript is then automatically generated (via the script `scripts/prepare/create_jobscript.py`) when you start a run.

7.4. Defining the parallelization layout

Parallelization is defined for each individual model, as described in the following subsections. The parallelization of the entire model system will then be derived from these individual settings automatically when you start a run.

7.4.1. Defining CCLM parallelization

CCLM uses a rectangular parallelization in X and Y direction, where X and Y are the zonal and meridional direction on the rotated lat-lon grid. The number of rectangles in each direction can be specified in the file `input/CCLM_???/INPUT_ORG`. This information will be automatically used by the script `/scripts/prepare/get_parallelization_layout.py`.

7.4.2. Defining MOM5 parallelization

MOM5 uses a rectangular decomposition defined in `input.nml`. A file named `mask_table` can be used to define which of the rectangles contain land points only. MOM5 comes with a tool to find out parallelization layouts which optimize the relative fraction of these land domains. This information will be automatically used by the script `/scripts/prepare/get_parallelization_layout.p`

We recommend to use the same layout for both ocean model and ice model.

7.5. Generating the exchange grid

To create the exchange grid

7.6. Creating the coupling namelists

7.6.1. Time stepping options

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! time stepping options - will be changed automatically !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
timestep      = *auto*,   ! coupling time step in seconds
num_timesteps = *auto*,

verbosity_level = 1,! 2 - debug, 1 - standard, 0 - error

name_atmos_model = 'CCLM_Eurocordex'
```

7.6.2. Input from bottom models

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
! Input from bottom models              !
! by (model,surface_type,runningnumber) !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!name_bottom_var_t(1,1:6,1) = 'FARE','none','none','none','none','none',
name_bottom_var_t(1,1:6,1) = 'FARE','FARE','FARE','FARE','FARE','FARE',
name_bottom_var_t(1,1:6,2) = 'TSUR','TSUR','TSUR','TSUR','TSUR','TSUR',
name_bottom_var_t(1,1:6,3) = 'FICE','FICE','FICE','FICE','FICE','FICE',
val_bottom_var_t (1,1:6,3) = 0.0, 1.0, 1.0, 1.0, 1.0, 1.0,
name_bottom_var_t(1,1:6,4) = 'ALBE','ALBE','ALBE','ALBE','ALBE','ALBE',
```

8. Running IOW_ESM

8.1. Starting an IOW_ESM run

How to start a run of the IOW ESM is described in detail in the `Readme.md` file.

8.2. Optimizing the runtime of the coupled model system

9. Postprocessing the data

9.1. Preparing the global settings

```
################################################################################
# STEP 1: Configure what will be plotted on a map (2D time averages)     #
################################################################################

# Define seasons for which time averages will be calculated.
# Note, the values in the dictionary must be valid input for the cdo operator
seasons = {
    "MAM" : "3,4,5",
    "JJA" : "6,7,8",
    "SON" : "9,10,11",
    "DJF" : "12,1,2",
    "year": "1,2,3,4,5,6,7,8,9,10,11,12"
}

# Percentiles are not supproted currently, leave empty!
percentiles = []

################################################################################
# STEP 2: Configure time series (1D spatial averages)                    #
################################################################################

# Define locations of stations for which you want to have time series.
# Note, longitudes and latitudes can be in decimal format or in degrees, minut
# Leave empty if no stations are desired.
stations = {
        "BY5" : {"lat" : "55.25", "lon" : "15.98"},
        "F9" : {"lat" : "64.71", "lon" : "22.07"},
        "SR5" : {"lat" : "61.08", "lon" : "19.58"},
        "BY31" : {"lat" : "58.58", "lon" : "18.23"}
}

# Define regions over which a spatial average will be performed.
# Note, longitudes and latitudes can be in decimal format or in degrees, minut
# Leave empty if no regions are desired.
regions = {
    "BALTIC_SEA" : {"lat-min" : "52.0", "lat-max" : "67.0", "lon-min" : "8.0",
    "BOTHNIAN_GULF" : {"lat-min" : "60.6", "lat-max" : "66.0", "lon-min" : "1
    "BALTIC_PROPER" : {"lat-min" : "53.0", "lat-max" : "60.6", "lon-min" : "1
    "BELTS" : {"lat-min" : "53.0", "lat-max" : "60.6", "lon-min" : "8.0", "lo
```

```
    "RIGA_FINLAND" : {"lat-min" : "56.5", "lat-max" : "60.6", "lon-min" : "23
}

# Define cdo operators which will be applied to the time series.
time_series_operators = ["-monmean", "-seasmean", "-ymonmean", "-yseasmean"]


##############################################################################
# STEP 3: Auxiliary steps (optional)                                         #
##############################################################################

# To customize the plotting you can import the PlotConfig class, see below fo
# If you leave that out, the plots will have gneric color maps and values ran
import sys
sys.path.append('../auxiliary')
from plot_config import PlotConfig


##############################################################################
# STEP 4: Define the variables for which the postprocessing is performed #
##############################################################################

# The keys of the dictionary must be the names of the variables, as they are
# Pass as values:
# "seasons" :                           a dictionary as given above
# "percentiles" :                       a list as given above
# "stations" :                          a dictionary as given above
# "regions" :                           a dictionary as given above
# "time-series-operators" :             a list as given above
# "plot-config" :                       a PlotConfig instance as given below

# "reference-file-pattern" :            a string containing the path pattern
# "reference-variable-name" :           a string containing the name of the va
#                                       (if "reference-file-pattern" is set th
# "reference-additional-operators" :    a string containing additional cdo ope
#                                       (if "reference-file-pattern" is set th
# "plot-config-anomaly" :               a PlotConfig instance for plotting the

variables = {
    "SST" : {       "seasons" : seasons,
                    "percentiles" : percentiles,
                    "stations" : stations,
                    "regions" : regions,
                    "time-series-operators" : time_series_operators,
```

```
                "plot-config" : PlotConfig("SST", lon_name = "xt", lat_name
                #"reference-file-pattern" : "/scratch/usr/mvkkarst/obs/Cope
                #"reference-variable-name" : "analysed_sst",
                #"reference-additional-operators" : "-chname,lon,xt -chname
                #"plot-config-anomaly" : PlotConfig("SST", lon_name = "xt",
            },
}
```

A. Details on individual fluxes

This section contains physical details about the fluxes that can be calculated in the flux calculator executable.

All fluxes are calculated positive upward, such that e.g. precipitation is always negative.

Fluxes may depend on the following variables:

code	symbol	description	provided by	unit
ALBE	α	shortwave surface albedo	ocean/land model	1
AMOI	$a_{moisture}$	turbulent diffusion coefficient for moisture	atmos model	1
AMOM	$a_{momentum}$	turbulent diffusion coefficient for momentum	atmos model	1
	c_d	specific heat capacity of dry air at constant pressure	1005.0	J/kg/K
	ΔH_v	latent heat of vaporization	2.501e6	J/kg
	ΔH_f	latent heat of freezing	0.334e6	J/kg
	ΔH_s	latent heat of sublimation	2.835e6	J/kg
FARE	f_{area}	area fraction of this bottom type (between 0 and 1)	ocean/land model	1
FICE	f_{ice}	ice area fraction (between 0 and 1)	ocean/land model	1
	g	graviational acceleration	9.81	m/s
PATM	p_a	pressure at lowest atmospheric level	atmos model	Pa
PSUR	p_s	surface pressure	atmos model	Pa
QATM	$q_{v,a}$	specific water vapor content (lowest atmospheric grid cell)	atmos model	kg/kg
	$q_{v,s}$	specific water vapor content (at sea / soil surface)	aux. variable	kg/kg
	R_d	dry air gas constant	287.05	J/kg/K
	R_v	water vapor gas constant	461.51	J/kg/K
	σ	Stefan-Boltzmann constant	5.67e-8	$W/m^2/K^4$
TATM	T_a	air temperature (lowest atmospheric grid cell)	atmos model	K
TSUR	T_s	surface temperature	ocean/land model	K
UATM	u_a	zonal velocity (lowest atmospheric grid cell)	atmos model	m/s
	$u_{min,evap}$	minimum velocity for CCLM evaporation calculation	0.01	m/s
VATM	v_a	meridional velocity (lowest atmospheric grid cell)	atmos model	m/s

A.1. Auxiliary variables for fluxes

A.1.1. specific water vapor content (at sea / soil surface)

This auxiliary variable ($q_{v,s}$, in kg/kg) describes the specific water vapor content in the air immediately above the surface, i.e. in the logarithmic boundary layer.

It is used further in the following routines:

	file	routine	variable
flux calculator	`flux_lib/mass/flux_mass_evap.F90`	`flux_mass_evap_cclm`	`specific_vapor_content_surface`

Here is how it is calculated:

COSMO-CLM routine for specific water vapor content at sea/ice surface

requires	f_{ice}, p_s, R_d, R_v, T_s	
source	COSMO-CLM subroutine `initialize_loop` (`lmorg.f90`)	
references	?, ?	
calculation	over water / over ice:	
	$\alpha = 17.2693882$ / $\alpha = 21.8745584$	1
	$T_2 = 35.86$ / $T_2 = 7.66$	K
	always:	
	$T_1 = 273.16$	K
	$p_0 = 610.78$	Pa
	$p_{sat} = p_0 \exp\left(\alpha \frac{T_s - T_1}{T_s - T_2}\right)$	Pa
	$q_{v,s} = \frac{\frac{R_d}{R_v} p_{sat}}{p_s - \left(1 - \frac{R_d}{R_v}\right) p_{sat}}$	kg/kg

A.2. Mass fluxes

A.2.1. Evaporation/condensation mass flux of water

The surface moisture flux (kg/m^2/s) is used further in the following routines:

	file	routine	variable
COSMO-CLM	`src_conv_tiedtke.f90`	`organize_conv_tiedtke`	`-qvsflx`
COSMO-CLM	`src_diagbudget.f90`	`organize_diagbudget`	`-qvsflx`
COSMO-CLM	`src_integrals.f90`	`check_qx_conservation`	`-qvsflx`
flux calculator	`flux_heat_latent.f90`	`flux_heat_latent_water`	`flux_mass_evap`
flux calculator	`flux_heat_latent.f90`	`flux_heat_latent_ice`	`flux_mass_evap`

Here is how it is calculated:

COSMO-CLM routine for evaporation/condensation mass flux

requires	$a_{moisture}$, p_s, $q_{v,a}$, $q_{v,s}$, R_d, R_v, T_s, u_a, $u_{min,evap}$, v_a	
source	COSMO-CLM subroutine `slow_tendencies`	(`slow_tendencies.f90`)
calculation	$\tilde{T} = T_s \left(1 + \left(\frac{R_v}{R_d} - 1\right) q_{v,s}\right)$	K
	$\lvert\vec{u}\rvert = \sqrt{u_a^2 + v_a^2}$	m/s
	$flux_{air} = a_{moisture} \max(\lvert\vec{u}\rvert, u_{min,evap}) \frac{p_s}{R_d \tilde{T}}$	kg/m^2/s
	flux_mass_evap $= flux_{air} \cdot (q_{v,s} - q_{v,a})$	kg/m^2/s
explanation	First we calculate the temperature $\tilde{T}$ at which dry air at the surface would show the same energy $p \cdot V$ as the moist air which is there now. We then calculate a mass flux of air coming into contact with the surface. Evaporation is then calculated assuming that this air adjusts its water vapor content to the one at the surface.	

A.3. Heat fluxes

A.3.1. Latent heat flux

The latent heat flux (W/m^2) is used further in the following routines:

	file	routine	variable
COSMO-CLM	`near_surface.f90`	`near_surface`	`-lhfl_s`
COSMO-CLM	`send_fld.f90`	`send_fld`	`-lhfl_s`
COSMO-CLM	`src_conv_ifs.f90`	`organize_conv_ifs`	`-lhfl_s`

Here is how the fluxes can be calculated:

COSMO-CLM routine for latent heat flux over water

requires	ΔH_v, `flux_mass_evap`	
source	COSMO-CLM subroutine `slow_tendencies`	(`slow_tendencies.f90`)
calculation	flux_heat_latent = $\Delta H_v \cdot$ `flux_mass_evap`	J/m^2/s

COSMO-CLM routine for latent heat flux over ice

requires	ΔH_s, `flux_mass_evap`	
source	COSMO-CLM subroutine `slow_tendencies`	(`slow_tendencies.f90`)
calculation	flux_heat_latent = $\Delta H_s \cdot$ `flux_mass_evap`	J/m^2/s

A.3.2. Sensible heat flux

The sensible heat flux (W/m^2) is used further in the following routines:

	file	routine	variable
COSMO-CLM	`near_surface.f90`	`near_surface`	`-shfl_s`
COSMO-CLM	`send_fld.f90`	`send_fld`	`-shfl_s`
COSMO-CLM	`src_conv_ifs.f90`	`organize_conv_ifs`	`-shfl_s`

Here is how the fluxes can be calculated:

COSMO-CLM routine for sensible heat flux

requires	$a_{moisture}$, p_a, p_s, $q_{v,s}$, R_d, R_v, T_s, u_a, $u_{min,evap}$, v_a	
source	COSMO-CLM subroutine `slow_tendencies`	(`slow_tendencies.f90`)
calculation	$\tilde{T} = T_s \left(1 + \left(\frac{R_v}{R_d} - 1\right) q_{v,s}\right)$	K
	$\lvert\vec{u}\rvert = \sqrt{u_a^2 + v_a^2}$	m/s
	$flux_{air} = a_{moisture} \max(\lvert\vec{u}\rvert, u_{min,evap}) \frac{p_s}{R_d \tilde{T}}$	$kg/m^2/s$
	$EF = \left(\frac{p_s}{p_a}\right)^{\frac{R_d}{cp_d}}$	1
	$flux\_mass\_evap = flux_{air} \cdot cp_d \cdot (T_g - T_a \cdot EF)$	$kg/m^2/s$
explanation	The first part until $flux_{air}$ is identical to the CCLM mass flux of evaporation. EF is the Exner function which calculates the ratio T/θ between local temperature at local pressure and potential temperature at a reference pressure. In the original routine two Exner factors were calculated for pressure in the lowest atmospheric level and surface pressure, each with a reference pressure of 1e5 Pa, and divided by each other, here the procedure is simplified. The Exner factor EF converts the temperature of the lowest atmospheric level T_a to temperature at the surface.	

A.4. Momentum fluxes

A.4.1. Upward flux of horizontal momentum

The upward flux of horizontal momentum (=shear stress, N/m^2) is used further in the following routines:

	file	routine	variable
COSMO-CLM	`near_surface.f90`	`near_surface`	`-umfl_s, -vmfl_s`
COSMO-CLM	`send_fld.f90`	`send_fld`	`-umfl_s, -vmfl_s`

Here is how it is calculated:

COSMO-CLM routine for eastward momentum flux

requires	$a_{momentum}$, p_s, $q_{v,s}$, R_d, R_v, T_s, u_a, v_a	
source	COSMO-CLM subroutine `slow_tendencies`	(`slow_tendencies.f90`)
calculation	$\tilde{T} = T_s \left(1 + \left(\frac{R_v}{R_d} - 1\right) q_{v,s}\right)$	K
	$\lvert\vec{u}\rvert = \sqrt{u_a^2 + v_a^2}$	m/s
	$flux_{air} = a_{momentum} \lvert\vec{u}\rvert \frac{p_s}{R_d \tilde{T}}$	kg/m^2/s
	flux_momentum_east = $-flux_{air} \cdot u_a$	kg/m/s^2=N/m^2
explanation	First we calculate the temperature $\tilde{T}$ at which dry air at the surface would show the same energy $p \cdot V$ as the moist air which is there now. We then calculate a mass flux of air coming into contact with the surface. Momentum transfer is then calculated assuming that this air passes all its horizontal momentum to the surface.	

COSMO-CLM routine for northward momentum flux

is completely analog to the previous one.

A.5. Radiation fluxes

Just like precipitation fluxes, radiation fluxes are calculated by the atmospheric model on the atmospheric grid. Unlike precipitation, they do however vary between different bottom model cells with a different temperature and albedo. We will outline here how this redistribution works.

Two types of radiation are distinguished by their wavelength, which are shortwave and longwave radiation. For each of these types, each bottom grid cell can have one albedo value, or possibly several different ones e.g. for different ice classes which share the area of the ocean grid cell. The average albedo for the atmospheric grid cell i, α_i^a, can therefore be calculated as follows:

$$\alpha^a_{i,\text{shortwave}} = \sum_{m=1}^{M}\sum_{j=1}^{B_m} \frac{\left|g_i^a \tilde{\cap} g_j^{b,m}\right|}{\left|g_i^a\right|} \sum_{c=1}^{C_m} \alpha^{b,m}_{c,\text{shortwave}} \cdot f_c^{b,m}(j) \tag{1}$$

$$\alpha^a_{i,\text{longwave}} = \sum_{m=1}^{M}\sum_{j=1}^{B_m} \frac{\left|g_i^a \tilde{\cap} g_j^{b,m}\right|}{\left|g_i^a\right|} \sum_{c=1}^{C_m} \alpha^{b,m}_{c,\text{longwave}} \cdot f_c^{b,m}(j) \tag{2}$$

Here, m is the index of the bottom model chosen, j is its grid cell index, the numerator in the fraction gives the area of intersection between atmospheric and bottom model grid cells where they are bidirectionally coupled, the denominator normalizes this area to the full area of the atmospheric grid cell, c is the class index (=ice class, soil type) used in bottom model m, $\alpha_c^{b,m}$ is the albedo defined for this class in the bottom model, and $f_c^{b,m}(j)$ is the area fraction of the bottom grid cell covered by this class.

Black-body radiation is emitted from the surface according to its temperature and longwave albedo and can be calculated with the help of the Stefan-Boltzmann constant σ:

$$\text{bbr} = (1 - \alpha_{\text{longwave}})\sigma T^4 \tag{3}$$

A similar horizontal averaging gives an average value for the atmospheric grid cell:

$$\text{bbr}^a_{i,\text{longwave}} = \sum_{m=1}^{M}\sum_{j=1}^{B_m} \frac{\left|g_i^a \tilde{\cap} g_j^{b,m}\right|}{\left|g_i^a\right|} \sum_{c=1}^{C_m} \left(1 - \alpha^{b,m}_{c,\text{longwave}}\right) f_c^{b,m}(j)\sigma \left(T^{b,m}(j)\right)^4 \tag{4}$$

From this value, we can calculate an effective radiative bottom temperature for the atmospheric grid cell:

$$T^a_{i,\text{longwave}} = \sqrt[4]{\frac{\text{bbr}^a_{i,\text{longwave}}}{(1 - \alpha^a_{i,\text{longwave}})\sigma}} \tag{5}$$

We use this temperature for the radiative flux calculation to make sure the emitted black-body radiation is exact.

After calculating the downward radiative fluxes (both shortwave and longwave) by the atmospheric model, we pass these to the flux calculator. The flux calculator will then calculate the upward radiative fluxes on the exchange grid, using the spatially detailed information on albedo

and surface temperature. This procedure ensures that also the average upward fluxes over the atmospheric grid cell are exactly identical between (a) the flux calculator and (b) the atmospheric model which calculate them independently.

In practice, it works like this:

1. Bottom models send surface temperature.
2. flux_calculator calculates and sends blackbody radiation.
3. CCLM receives surface temperature and blackbody radiation in
   `lmorg -> initialize_loop -> receive_fld`
4. CCLM calculates all radiation components in
   `lmorg -> organize_physics('compute',...)`
   `-> organize_radiation`
5. CCLM sends radiation fluxes and other variables in
   `lmorg -> organize_physics('compute',...)`
   `-> communicate_with_flux_calculator`
6. flux_calculator calculates and sends all other fluxes.
7. CCLM receives these fluxes in the same routine.
8. Bottom models receive all fluxes.

A.5.1. Blackbody radiation

The upward flux of blackbody radiation (=thermally emitted longwave radiation, W/m^2) is used further in the following routines:

file	routine	variable

Here is how it is calculated:

Calculation after Stefan-Boltzmann law

requires	σ, T_s
source	any textbook
calculation	flux_radiation_blackbody = σT_s^4 $W/m^2/K^4$
explanation	This is the well-known Stefan-Boltzmann law, it assumes that the surface acts as a perfect black body in the long-wavelength band.

B. Previous way of coupling

B.1. Sandra's approach

B.1.1. OASIS3-MCT routines in COSMO-CLM

This is where the OASIS3-MCT coupling routines were called in COSMO-CLM in the previous version by Sandra-Esther Brunnabend (largely inherited from the version by Ha Ho-Hagemann at HZG):

```
lmorg                              ( lmorg.f90        )
  organize_setup                   ( src_setup.f90    )
    init_environment               ( environment.f90  )
      oas_cos_init                 ( oas_cos_init.f90 )
        >prism_init_comp_proto      --> initialize the PRISM system
        >prism_get_localcomp_proto --> get MPI communicator
  read_namelist_oasis              ( oas_cos_define.f90 )
    input_oasisctl                 ( oas_cos_define.f90 )
  oas_cos_define                   ( oas_cos_define.f90 )
    >prism_start_grids_writing      --> write grids to external files
    >prism_write_grid
    >prism_write_corner
    >prism_write_mask
    >prism_write_area
    >prism_terminate_grids_writing
    >prism_def_partition_proto      --> offset and size of this PE's
                                        rectangle
    >prism_def_var_proto            --> names of variables to be sent
                                        and received
    >prism_enddef_proto
  initialize_loop                  ( lmorg.f90 )
    receive_fld                    ( receive_fld.f90 )
      oas_cos_rcv                  ( oas_cos_rcv.f90 )
        >prism_get_proto            --> receive fields from coupler
  send_fld                         ( send_fld.f90 )
    oas_cos_snd                    ( oas_cos_snd.f90 )
      >prism_put_proto              --> send fields to coupler
  final_environment                ( environment.f90 )
    oas_cos_finalize               ( oas_cos_finalize.f90 )
      >prism_terminate_proto
```

B.1.2. Radiation flux calculation in CCLM

Radiation is calculated in subroutine `organize_radiation` in `src_radiation.F90`. The following values are used:

- Shortwave albedo: Since `ytype_oce=='momBS'`, there are defined lon-lat boxes in which the MOM5 model ice fraction `fr_ice(i,j)` is used to have an average albedo between ice and ocean. Outside these, each water cell is either ice or ocean depending on the value of `t_test`. For open water we use the value `csalb(9)`=0.07 and for ice we use `csalb(10)`=0.70 as defined in `data_soil.F90`.
- Longwave albedo: Since `lemiss==FALSE`, we use the value `ctalb`=0.004 defined in `data_soil.F90`.
- Temperature for black-body radiation: `t_g(i,j)` is used and saved in `zti(i,j,ke1)`. The array `zti` stores the temperatures at the interfaces between vertical layers.

The following fields are communicated to MOM5:

```
Name of field
|        Name in organize_radiation()
|        |       Argument nr. in fesft()
|        |       |  Name in fesft()
         zflt_s  42 pflt_s  (i,j)   ! thermal net surface flux
         zfls_s  43 pfls_s  (i,j)   ! shortwave net surface flux
         zflsdir 44 pflsdir (i,j,k) ! solar direct downward radiative
lwd_s    zfltd   45 pfltd   (i,j)   ! thermal downward
lwu_s    zfltu   46 pfltu   (i,j)   ! thermal upward
swdifd_s zflsd   47 pflsd   (i,j)   ! shortwave diffusive downward
swdifu_s zflsu   48 pflsu   (i,j)   ! shortwave diffusive upward
swdir_s  zflsp   49 pflsp   (i,j)   ! shortwave direct downward
```