Building and Running EC-Earth 3

A short guide

Table of Contents

Introduction

EC-Earth[1] is a global coupled climate model, which integrates a number of component models in order to simulate the earth system. It is developed by a consortium of European research institutions, which collaborate in the development of a new Earth System Model. The goal of EC-Earth is to build a fully coupled Atmosphere-Ocean-Land-Biosphere model, usable from seasonal to decadal climate prediction and climate projections.

EC-Earth 3 is the most recent model generation and the one described here. The intention of this guide is to document mostly technical aspects that are of importance when getting started with EC-Earth 3.

In order to gather the latest and most relevant information about EC-Earth 3 development in one place, a Development Portal has been established. This is a Web-based service, which can be reached via the following URL:

```
https://dev.ec-earth.org
```

The site includes further documentation, issue tracking facilities, and access to the version control system. It is strongly recommended to refer to the Development Portal first when any concerns or requests need to be addressed.

Preparations

Technical Prerequisites

A number of tools and libraries are needed to configure and build EC-Earth 3:

Prerequisite	Notes
Compiler/Languages	
GNU make	GNU make version 3.81 or more is needed to build IFS
Fortran	A Fortran 77/90/95 compliant compiler with preprocessing capabilities is needed.
C/C preprocessor	
Python	The half-automatic build configuration tool ec-conf is tested with Python v2.4.3 and above. It will, however, not work with Python v3.0+
Bash	Used by FCM
Perl	Used by FCM
Libraries	
MPI	
BLAS/LAPACK	

1) W. Hazeleger et al (2010) *EC-Earth: a seamless Earth system prediction approach in action.* Bull Am Meterol Soc 91:1357–1363. doi:10.1175/2010BAMS2877.1

Prerequisite	Notes
NetCDF	
GRIB API	Needed for IFS GRIB I/O. Version 1.12 has been tested but others may work as well[2]. Download from `https://software.ecmwf.int/wiki/display/GRIB/Releases/`
GRIBEX	Needed for IFS GRIB I/O. Tested with version 370. GRIBEX used to be provided at ECMWF, but is no longer available from their download site. A copy of version 370 is located at the EC-Earth 3 SVN server at `/ecearth3/vendor/gribex`.
HDF4	Required for TM5-MP
HDF5	Needed for netCDF4, which is needed for TM5. Should be compiled with parallel IO enabled.
Other tools	
makedepf90[3]	Needed, if dependency files for the IFS code need to be created (because they where removed or invalidated by a change of source files). This tool is provided both pre-compiled (for Linux) and as source code in the EC-Earth package.
TkInter (Python module)	Needed if the graphical interface of the ec-conf tool is used.
GNU date (64-bit version)	Needed for full run-script functionality.
Cmake	Needed to compile LPJ-Guess

Getting the Sources

EC-Earth 3 is distributed in source form, hence, it must be configured for the platform in use and built (compiled and linked) by the user. The principal distribution method for the source code is to access the Subversion (SVN) server, which is part of the EC-Earth 3 Development Portal. The SVN server is located at the URL

```
https://svn.ec-earth.org
```

In order to understand the SVN repository structure in more detail, please refer to the Wiki article at

```
https://dev.ec-earth.org/projects/ecearth3/wiki/ \
                                    EC-EARTH_3_Version_control_strategy
```

Note that a backslash (\) has been used above to indicate that the line has been split in this guide for better readability.

The following instructions assume that the most recent status in the development of EC-Earth 3 (known as trunk) is to be accessed. For other purposes, e.g. for accessing a released version, the URL's have to be adjusted accordingly. For an up-to-date listing of available releases, check

2) As of May 2017, version 1.17.0 at ECMWF prevents compilation of LPJ-Guess.
3) Available at `http://personal.inet.fi/private/erikedelmann/makedepf90` under the GNU General Public License version 2.

```
> svn ls https://svn.ec-earth.org/ecearth3/tags
```

There is a certain freedom in how to lay out the source code and runtime directories in the user's file system. Therefore, the following instructions may be modified, provided that the changes are consistent.

The following command provides a complete structure of the EC-Earth 3 source, runtime, and documentation directories:

```
> svn checkout --username USER_NAME --password PASSWORD \
           https://svn.ec-earth.org/ecearth3/trunk ECEARTH3_BASE_DIR
```

Note that the highlighted terms have to be replaced by the Development Portal account details (`USER_NAME`, `PASSWORD`) and a user chosen directory name (`ECEARTH3_BASE_DIR`), which will hold all of the EC-Earth 3 components.

For the remainder of this guide, it will be assumed that the source code was obtained by the above procedure, thus, the directory structure is expected to follow the mentioned layout.

Please refer to the documentation of the SVN client that is being used about how the user name and password information is handled. It might be possible to store the account details, thus avoiding specifying them repeatedly. However, be aware of security implications.

Building

Build Configuration

Once the source code of EC-Earth 3 is in place, the systems needs to be configured for building. The recommended method for build configuration is to use the ec-conf tool, in fact, this is the only method described in this guide. Manual configuration is, however, possible by editing the ec-conf configuration files by hand, even though this is an error-prone procedure.

The ec-conf tool is provided in the `ECEARTH3_BASE_DIR/sources/util/ec-conf` subdirectory and it comes with it's own documentation in `ECEARTH3_BASE_DIR/doc`. The usage of ec-conf is only very briefly explained here, as far as it is needed to understand the EC-Earth 3 build configuration process.

All configurable parameters are collectively stored in an XML data base file, which is to be adapted by the user according to the computing platform and build environment. For a detailed description of the XML file syntax, please refer to the ec-conf user guide. Once the XML data base file has been adapted, it is subsequently used by the ec-conf tool to create the required configuration files for all components. Hence, from a user's perspective, a *single place, single syntax configuration* is achieved.

Summarising, the usage of ec-conf comprises two stages, namely

- Adapting the configuration parameter in the XML data base file, and
- Running the ec-conf tool to create the actual configuration files.

Any editor that handles plain-text files can be used during the first step. Thereafter, the second step facilitates the command line interface of ec-conf. Alternatively, both steps can be dealt with in one go by utilising the ec-conf graphical user interface (GUI) by using the `--gui` option. Please refer to the ec-conf user guide for a detailed description of entire process.

When the sources are obtained according to the instructions in the previous chapter, and XML data base file is provided at

```
ECEARTH3_BASE_DIR/sources/config-build.xml
```

as a basis for build configuration. This file needs to be complemented with a `Platform` section suitable for the user's computing platform and the configuration parameters must be adjusted for the particular environment. It is strongly recommended to add new `Platform` sections when needed rather than re-using existing ones. This is to make sure that platform-dependent configurations are persistently documented.

Assuming that the XML data base file has been adjusted (and that `ECEARTH3_BASE_DIR` is the current directory), the ec-conf command line interface is invoked as

```
> cd sources
> ec-conf --platform PLATFORM config-build.xml
```

provided that the ec-conf command is present in the search path. The `PLATFORM` argument refers to the corresponding `Platform` section in the XML file. Upon completion of this command, all configuration files are created in their proper places. It is important to repeat that step every time the configuration is changed. Note that ec-conf may display warnings about empty parameter values. As long as the corresponding parameters are deliberately left without values (e.g. because a certain platform does not support or need a specific parameter), these warnings can be ignored.

Compiling

Assuming that the build configuration has been carried out with the ec-conf tool, the compilation of the component models is the next step. As both IFS and NEMO depend on OASIS, the coupler software has to be compiled first. Furthermore, NEMO depends on XIOS, so XIOS has to be compiled before NEMO. An example compilation sequence is OASIS, XIOS, NEMO, IFS, Runoff-mapper.

For forced atmosphere experiments (i.e. IFS-standalone runs), the AMIP-reader has to be compiled. This can be done after the other components.

For the Earth System Model (ESM), TM5-MP and/or LPJ-Guess must be compiled. This can be done in any order, but after OASIS.

Compiling Oasis

Provided that `ECEARTH3_BASE_DIR/sources` is the current directory, the build process is started from the build directory of OASIS:

```
> cd oasis3-mct
> cd util/make_dir
```

where compilation of the libraries and the executable is initiated by the command

```
> make BUILD_ARCH=ecconf -f TopMakefileOasis3

--> Using BUILD_ARCH=ecconf
--> Reading configuration from

[... more output follows ...]

make[1]: Leaving directory ...
```

Upon successful completion of the process (i.e. when no errors are displayed), the OASIS libraries, module files, and the executable can be found below the ECEARTH3_BASE_DIR/sources/oasis3-mct/ecconf directory.

Compiling XIOS

XIOS (short for XML-I/O-Server) is an I/O management software for climate models. It handles output of diagnostics as well as temporal and spatial post-processing operations. XIOS is currently used by the ocean component, NEMO, of EC-Earth, with the intention to extent it's use to other component models in the future.

Two major versions of XIOS are currently available: XIOS1 and XIOS2. The NEMO ocean model in EC-Earth uses XIOS2, which is provided as part of the EC-Earth source tree.

Provided that ECEARTH3_BASE_DIR/sources is the current directory, XIOS2 is compiled by the following commands:

```
> cd xios-2

> ./make_xios --arch ecconf --use_oasis oasis3_mct <OPTIONAL_ARGUMENTS>

Optional arguments:

--netcdf_lib netcdf4_seq     - If XIOS is only used in sequential mode
--full                       - To recompile everything
--job <N>                    - For parallel make with N jobs
```

It is recommended to compile XIOS2 with a parallel NetCDF library (by not using the `--netcdf_lib option` or specifying `--netcdf_lib netcdf4_par`). Using parallel NetCDF allows the XIOS servers to run in parallel mode, which may be needed on machines with little memory per node. The parallel mode may also be beneficial for the computational performance of the I/O subsystem.

Compiling NEMO

NEMO (from version 3.3 onwards) uses the Flexible Configuration Management (FCM)[4] for compilation. The FCM configuration file for NEMO has been adapted to support ec-conf and it is recommended to use this tool to build NEMO within EC-Earth 3.

The main script for compiling NEMO is located in the CONFIG subdirectory

```
> cd nemo-3.6
> cd CONFIG
```

4) http://cms.ncas.ac.uk/index.php/fcm

and it is used like this:

```
> ./makenemo -n ORCA1L75_LIM3 -m ecconf -j NUM_PROC
```

By applying the above command line, it is assumed that ec-conf has been successfully run and that the default configuration (comprising the ORCA1 ocean grid with 75 vertical levels and the LIM3 sea ice model) is to be built. Other configurations are available, which can be listed (among other information) by

```
> ./makenemo -h

Usage  : makenemo [-h] [-n name] [-m arch] [-d dir1 dir2] [-r conf] [-j No]

[... more output follows ...]

Available configurations :

ORCA1L75_LIM3 [...]
ORCA1L75_LIM3_standalone [...]
ORCA025L75_LIM3 [...]
ORCA025L75_LIM3_standalone [...]
ORCA1L75_LIM3_PISCES_standalone [...]
ORCA1L75_OFF_PISCES [...]
```

Note that the compilation of NEMO (when calling the `./makenemo` script as shown before), it is possible to specify the number of parallel compilation processes with the `-j NUM_PROC` command line argument.

When compilation has succeeded, a line similar to

```
Build command finished on <SOME_DATE>.
```

should appear and the NEMO executable, `nemo.exe`, is located in the `ORCA1L75_LIM3/BLD/bin` subdirectory (or a similar one corresponding to the chosen configuration). Note that it is possible to compile and keep more than one configuration of NEMO at a time. In fact, this is supported by the EC-Earth 3 run scripts.

Compiling IFS

The IFS source files and build scripts are located in the ifs-36r4 subdirectory of `ECEARTH3_BASE_DIR/sources`:

```
> cd ifs-36r4
```

All makefiles have been configured by ec-conf, hence, the build step can be started immediately. As a first step, all IFS libraries are compiled. The build process supports parallel make, which is activated by the `-j` command line argument. The number of parallel compilation processes, which can be used efficiently, has to be established experimentally on the build platform, but it is probably less than ten. The make command to compile the libraries reads

```
> make BUILD_ARCH=ecconf -j NUM_PROC lib
```

After all libraries are built successfully, the IFS executable (the ifs-master) has to be linked. There is no advantage in using parallel make in that step as only one compilation instance is involved:

```
> make BUILD_ARCH=ecconf master
```

When the linking process has completed, the IFS executable will be found in the `bin` subdirectory.

NOTE: Due to the current structure of the makefiles, it is not possible to specify both the `lib` target and the `master` target at the same time in the make command line.

Compiling the Runoff-mapper

The Runoff-mapper is a small program that splits the run-off from IFS into a run-off and a calving contribution, and remaps the fields onto the NEMO grid. It is compiled by the following commands:

```
> cd runoff-mapper/src
> make
```

Compiling the AMIP-reader

The AMIP-reader is used for IFS-standalone experiments and provides IFS with forcing data via OASIS. The compilation of the AMIP-reader is done with:

```
> cd amip-forcing/src
> make
```

Compiling TM5-MP

The sources and the set-up script to compile TM5-MP are in the tm5mp subdirectory of `ECEARTH3_BASE_DIR`/sources:

```
> cd tm5mp
```

By running ec-conf on the config-build.xml, the file `ecconfig-ecearth3.rc` is created. It is the input configuration file for TM5 set-up script, which you invoke as follows:

```
> ./setup_tm5 -j NUM_PROC ecconfig-ecearth3.rc
```

A list of available options can be listed with:

```
> ./setup_tm5 -h
```

Of importance for EC-Earth are `-n` (or `--new`) also known as `realclean` in many programs to recompile the entire code, and `-c` (or `--clean`) to recompile only part of the code (the generic code for reading HDF/netCDF files is left as is).

When compilation succeeds, the output ends up with lines like those:

```
[INFO    ] For first glance on settings and results:
[INFO    ]
[INFO    ]   # run diadem postprocessor:
[INFO    ]   ./tools/diadem/py/diadem ecconfig-ecearth3.rc
[INFO    ]
[INFO    ] End of script at 2014-09-17 09:02:34 .
[INFO    ]
```

Two different executables can be compiled: one with the NOx-OH-HC-aerosols atmospheric chemistry activated, the other concerned only with CO2. The switch is done in the `config-build.xml` with the parameter "TM5_CO2_ONLY" of your platform. Both executables can be compiled without affecting each other: just modify, parse the config-build.xml file and call setup_tm5 again. And the executables are found in:

```
tm5mp/build/src/appl-tm5.x
tm5mp/build-co2/src/appl-tm5-co2.x
```

Compiling LPJ-Guess

The sources of LPJ-Guess are in the `lpjg` subdirectory of `ECEARTH3_BASE_DIR/sources`:

```
> cd lpjg/build
```

When running `ec-conf`, the file `lpjg/CMakeLists.txt` is created. It is the configuration file `cmake`, which you invoke as follows:

```
> cmake ..
> make
```

When compilation succeeds, the output ends up with these lines:

```
[100%] Building CXX object CMakeFiles/guess.dir/command_line_version/main.cpp.o
Linking CXX executable guess
[100%] Built target guess
```

And the executable is:

```
lpjg/build/guess
```

With this, all the EC-Earth components have been compiled and the model is (almost) ready to run.

System-wide EC-Earth 3 Installations

It is rather easy to install and maintain a system-wide EC-Earth 3 installation, because the model source code and runtime environment are largely independent of each other. System-wide installations can make it easier for new users to get started (because the building step is skipped) and can provide a higher level of consistency and efficiency for coordinated experiments.

To install EC-Earth system-wide, the model source code is downloaded from the SVN server to a central place, `ECEARTH3_BASE_DIR`, and built as described in the previous sections of this document. Users need to have read-access to `ECEARTH3_BASE_DIR`.

Further more, the initial data sets are downloaded to another central place, `INI_DATA_DIR`, as described in the next section, “Providing EC-Earth 3 Experiments with Initial Data”. The users to run EC-Earth experiments need obviously also read access to `INI_DATA_DIR`.

It is an advantage for users if both `ECEARTH_SRC_DIR` (derived from `ECEARTH3_BASE_DIR`) and `INI_DATA_DIR` are correctly specified in the SVN repository ver-

sion of the `config-run.xml` file for the particular platform. If that is the case, the user does not need to modify these entries after downloading the runtime environment from the SVN server.

Once the model source code and initial data is in place, users just have to check out the runtime environment from the SVN server:

```
> svn checkout --username USER_NAME --password PASSWORD \
           https://svn.ec-earth.org/ecearth3/trunk/runtime/classic \
           EXPERIMENT_DIR
```

Thus, a copy of the `classic` runtime environment of the trunk is created in the `EXPERIMENT_DIR` directory. Note that this is much faster than checking out the complete source code of EC-Earth. With the runtime environment in place (and assuming that `ECEARTH_SRC_DIR` and `INI_DATA_DIR` have correct values for the particular platform), the user is ready to configure and run experiments as explained later in section “Running EC-Earth”.

It is, of course, possible to install another version of EC-Earth than just the trunk that is used in the example above. However, it is important that the versions of the source code in `ECEARTH3_BASE_DIR`, the initial data in `INI_DATA_DIR`, and the user's runtime environment match. There may be changes in the EC-Earth source code that lead to inconsistencies in the runtime environment across versions (such as changed namelists). It is the responsibility of the maintainer of the the system-wide installation to make sure compatible versions are used.

Providing EC-Earth 3 Experiments with Initial Data

EC-Earth 3 experiments need a number of data files for configuration, initialisation, and forcing. These files are not part of the EC-Earth source code and, thus, cannot be downloaded from the SVN server that provides the source code. The necessary files are bundled in data sets and provided by other means than SVN.

To allow for validation of the data, each file is checksummed and the checksums are stored in the SVN repository. This facilitates data provenance, both to check the validity and to keep a historical records of the file changes. The checksum files are located in the `ECEARTH3_BASE_DIR/runtime/datacheck` directory, and they can be used to make sure the data is consistent with the current version of the code:

```
> cd INI_DATA_DIR
> md5sum -c ECEARTH3_BASE_DIR/runtime/datacheck/ece-data-base.md5
```

The above command checks the validity of the `ece-data-base.md5` data set, assuming the data has been downloaded to `INI_DATA_DIR`.

Beside the actual checksums, the checksum files provide also a maintainer of the corresponding data set as well as a download URL:

```
> head ece-data-base.md5

# Checksums (md5sum) for ece-data-base.tgz
# Verify files with "md5sum -c ece-data-base.md5" while in the inidata \
```

```
directory
#
# Download URL: \
http://exporter.nsc.liu.se/9e71db1b1d1541a189a5d2bf8ab046d0/ece-data-base.tgz
# Download URL: \
rsync://exporter.nsc.liu.se/9e71db1b1d1541a189a5d2bf8ab046d0/ece-data-base.tgz
# Maintainer:   uwe.fladrich@smhi.se
#
d390e563f044e53e264109767615ad33  ifs/rtables/rtable_2106
d390e563f044e53e264109767615ad33  ifs/rtables/rtablel_2159
46266ff4cf07500a53b5d9a1b282a2ff  ifs/rtables/rtable_12213
[...]
```

Note that the above output is just given as an example. Do not assume that the actual information is correct for times other than at this writing.

Running EC-Earth 3

Setting up and running experiments with EC-Earth is a pretty complex task. Part of the difficulty stems from the great variety of experiment configurations and computational platforms. It is next to impossible to provide a runtime environment that can handle all desirable configurations, while at the same time maintaining a reasonable complexity. Thus, EC-Earth comes with a runtime environment that provides support for a set of basic experiment types and platforms. EC-Earth users should be prepared to use the provided scripts as a basis for extensions to their own needs.

The EC-Earth runtime environment comes in two flavours: One (called `classic`) with scripts that are supposed to be manually submitted to job schedulers, and another (called `autosubmit`) that will work well with the Autosubmit[5] work-flow manager.

The description below corresponds to the `classic` runtime environment.

Run Configuration with ec-conf

In order to run EC-Earth 3, run scripts have to be prepared to reflect the current platform and the experiment at hand. The recommended way of doing this is by using the ec-conf tool, much in the same way as when building EC-Earth 3. The ec-conf tool is provided in the `ECEARTH3_BASE_DIR/sources/util/ec-conf` sub-directory. Because ec-conf has its own documentation in the `ECEARTH3_BASE_DIR/doc` directory only the usage of the tool, not the tool itself, is described here.

Setup Prerequisites for Using ec-conf

Preparing the run scripts with the ec-conf tool on a specific platform requires the user to provide:

- A platform dependent template file, which provides the shell functions `configure()`, `launch()`, and `finalise()`. This functions are required to handle, respectively, any

5) http://autosubmit.readthedocs.io

platform dependent configuration, launch mechanism for an MPMD MPI job, and, if necessary, post run operations.

- All configurable parameters, e.g., platform dependent run time paths and specific experiment settings. These have to be stored in an XML data base file for the use with ec-conf.

The first step is needed only once for each platform. Existing template files are located (again assuming the earlier defined directory structure) at

```
ECEARTH3_BASE_DIR/runtime/classic/platform/PLATFORM.cfg.tmpl
```

and can server as templates for new platforms.

In the second step, some of the settings in the XML data base file are platform dependent (`RUN_DIR`, `PROC_PER_NODE`, ...) and some of them need to be updated according to each experiment (`EXP_NAME`, `RUN_START_DATE`, ...).

An example XML database file for the run scripts is provided at

```
ECEARTH3_BASE_DIR/runtime/classic/config-run.xml
```

This file, together with the existing template files and the documentation on the XML database file structure in the separate ec-conf documentation, serves as a guide when setting up EC-Earth 3 on a new platform.

Generating Run-scripts with the ec-conf Command Line Interface

Assuming the user has a template file for a given platform and has filled out all the configurable parameters in the XML database file in his/her favourite text editor, the actual run scripts are created by issuing,

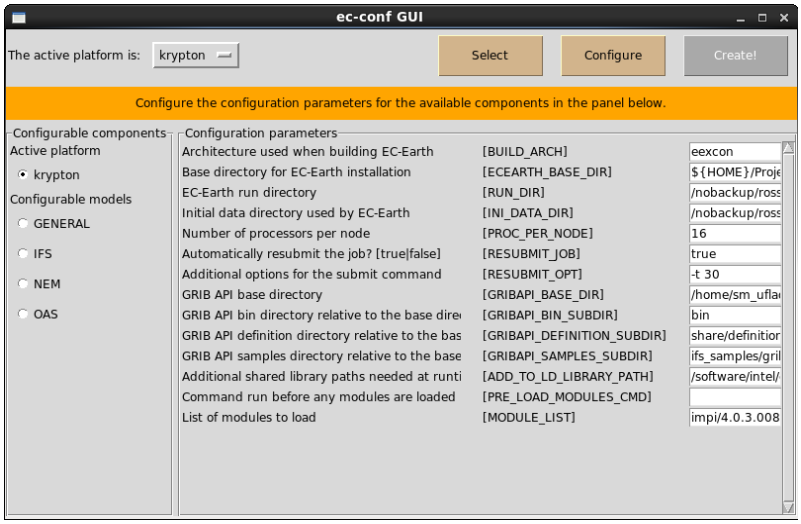
```
> ec-conf --platform PLATFORM RUN_CONFIG_FILE
```

As when building EC-Earth 3, `PLATFORM` refers to the appropriate platform section in the XML file in use and `RUN_CONFIG_FILE` is the XML database file. This command will create run scripts, such as,

- `ece-ifs+nemo.sh` Run-script for a coupled experiment
- `ece-ifs.sh` Run-script for an atmosphere-only (IFS) experiment
- `ece-nemo.sh` Run-script for an ocean-only (NEMO) experiment
- `ece-esm.sh` Run-script for an Earth System Models experiment

Other scripts are created as well, depending on the exact content of the ec-conf XML file.

Generating run-scripts with the ec-conf Graphical User Interface



The ec-conf GUI while being used for run-script generation

Alternatively, the run-scripts can be prepared with the Graphical User Interface (GUI) of ec-conf, which provides all of the functionality of the Command Line Interface. In fact, the GUI might turn out to be preferable when used to create run-scripts in a more interactive way. It is, for example, possible to generate a number of run-scripts, which differ in in just a few, specific aspects, such as the start date of the experiments. Thus, the run-scripts can later be used in a batch-like fashion without the need for manually copying and modifications.

The ec-conf GUI is launched with the command

```
> ec-conf --gui RUN_CONFIG_FILE
```

The ec-conf GUI also allows the user to interactively update the XML database file, save it for future use and generate the run-scripts. See the ec-conf documentation for details on the GUI usage.

General ec-conf Hints

Some general advice for using ec-conf include:

- Make sure ec-conf is in your search path
- If you are using relative paths in the `RUN_CONFIG_FILE`, make sure they are consistent with your current working directory used when running ec-conf
- ec-conf warnings are prefixed `*WW*`, and output to `stderr`. Check if the warning is relevant to your particular case, if not, it can be ignored
- ec-conf errors are prefixed `*EE*` and cannot be ignored as no run-scripts are generated

Running EC-Earth 3 Experiments

Running EC-Earth 3 on a specific platform

The EC-Earth 3 Wiki includes instructions about how to submit the generated run-scripts on a number of platforms. The Wiki also points out where the default initial data files can be found for that platform. Users who are setting up EC-Earth 3 on new platforms are encouraged to add corresponding information to the Wiki.

Further configuration

There are three levels of configuration available for an experiment. The principal user interface is the `config-run.xml` described in the previous section. Additional settings are available through the `config` variable found at the top of the scripts generated when calling `ec-conf`. This level of configuration is mainly used for switching off/on various component of the Earth System Model (see "Running an ESM model" section hereafter) or options like stochastic physics. Finally, experienced users can also modify the various namelists to their need. You are referred to the models respective documentation for further details.

Running IFS standalone

TODO: Check is this paragraph is up-to-date (it is probably not)

The atmospheric component of EC-Earth 3 can be run as a standalone experiment, usually no special preparations are necessary for launching such experiment. The run scrip `ece-ifs.sh`, prepared as described above, will start the job and use the necessary forcing files which are packaged together with the initial data files for each resolution. See the Wiki for details on how to access EC-Earth 3 initial files. **NB:** By default forcing over sea includes prescribed SST and sea-ice cover. If there is a need to prescribe sea-ice temperature as well it is necessary to,

1. Set `LRDISTL1 = TRUE` in the namelist `NAMMCC`
2. Update the ICMSEAINIT-file according to

Without prescribed sea-ice temperature					
paramId	shortName	edition	centre	dataDate	level
34	sst	1	ecmf	19791201	0
31	ci	1	ecmf	19791201	0
34	sst	1	ecmf	19800101	0
31	ci	1	ecmf	19800101	0
...	and	so	on	...	

With prescribed sea-ice temperature					
paramId	shortName	edition	centre	dataDate	level
34	sst	1	ecmf	19791201	0
31	ci	1	ecmf	19791201	0
35	istl1	1	ecmf	19791201	0
34	sst	1	ecmf	19800101	0
31	ci	1	ecmf	19800101	0
35	istl1	1	ecmf	19800101	0
...	and	so	on	...	

Running NEMO standalone

TODO: This is no longer relevant

~~The oceanographic component of EC-Earth 3 can be run as a standalone experiment. To do so it is necessary to compile the NEMO binary without the keys key_coupled and key_oasis3. These keys are specified in the respective NEMO compile key configuration file, e.g.,~~

```
~~ECEARTH3_BASE_DIR/nemo-3.6/CONFIG/ORCA1L75_LIM3/cpp_ORCA1L75_LIM3.fcm~~
```

~~Simply remove the two keys from the above file and (re-)compile NEMO along the directions given in the above sections. The run scrip run-oce.sh, prepared as described above, will start the job. Note that~~

- ~~forcing files for running NEMO standalone has to be downloaded separately, see the Wiki for details on how to access NEMO standalone forcing files~~
- ~~some NEMO configurations may not have all the necessary files for running in standalone mode~~

Running an ESM model

The `ece-esm.sh` script lets you run EC-Earth in at least eight different configurations. To switch between them, just change the **`config`** variable found at the top of the script. This variable lists the components to run. Options can be added to each component with a colon ":". For example, the following configuration will let you run IFS with the AMIP-reader and LPJ-Guess:

```
config="ifs amip lpjg:fdbck"
```

Note that the fdbck option is added to LPJ-Guess. Without it, IFS would drive LPJ-Guess, but not receive any feedback.

The following configurations will work with forced SST and sea ice cover:

config	**Comment**
"ifs amip"	"forced GCM": IFS + AMIP reader
"ifs amip lpjg:fdbck tm5:co2"	"C-cycle" : forced GCM + LPJ-Guess + TM5
"ifs amip lpjg:fdbck"	"Veg" : forced GCM + LPJ-Guess
"ifs amip tm5:chem,o3,ch4,aero"	"AerChem" : forced GCM + TM5

To couple the ocean model NEMO instead of reading sst/sea-ice forcings, replace '`amip`' with '`nemo lim3 rnfmapper xios:detached oasis`' in the table, which brings us to a total of eight configurations.

The following table gives a brief description of all available components and their options.

Component:option(s)	Comment
ifs	run IFS, always needed
amip	SST and sea-ice concentration reader. Must be used if and only if not running NEMO
nemo lim3	run NEMO with embedded LIM3 ice-model
tm5:chem	IFS drives the full chemistry version of TM5-MP
tm5:co2	IFS drives the CO2-only version of TM5-MP
tm5:o3,ch4,aero	TM5-MP feeds back O3, CH4 and Aerosols concentrations and optical properties to IFS. Effective only if tm5:chem is used.
lpjg	IFS drives LPJ-Guess (soil moisture and temperature, precipitation, snow,..)
lpjg:fdbck	IFS drives LPJ-Guess, and LPJ-Guess sends back LAIs, vegetation type and fraction to IFS
rnfmapper	runoff mapper between IFS and NEMO. Required if they are coupled.
xios:detached	xios output server used by nemo.
oasis	used by xios to indicate that nemo is coupled to ifs.
ifs:atmnudg	run IFS with nudging. See `IFS_atmospheric_nudging` on the wiki.
ifs:sppt	run IFS with stochastic physics. See `SPPT_stochastic_perturbations_for_IFS` on the wiki.
nemo:start_from_restart	let you use NEMO restart for the first leg of an integration

When coupling TM5, either `tm5:chem` or `tm5:co2` should be used. Any of the `o3,ch4,aero` feedback on IFS can be switched off by removing it.

Saving Initial Conditions during a run

Background information on Initial Conditions

Initial Conditions (ICs) are files which are used to initialize the components of the EC-Earth model at specific states. IC files are used, for example,. for seasonal prediction experiments which are initialized at various start dates with given atmosphere (IFS), ocean (NEMO) and sea-ice (LIM) states. The Initial Data files contain ICs for IFS and the OASIS coupler for a number of years, whereas NEMO is initialized by default from climatology. Initial conditions for other components (e.g. TM5, LPJ-Guess) are also available.

Except for IFS, ICs are identical to restart files which are used to save the model state at the end of a leg, and read at the start of the next leg. In the case of IFS, restart files must be used on the same machine and processor configuration that they were generated, therefore they are not portable. They cannot be changed to generate ensemble member using small perturbations.

How to save Initial Conditions during a run

The save_ic tool of EC-Earth can be used to generate initial conditions (ICs) for IFS, NEMO and the OASIS coupler files for IFS and NEMO at any time during an experiment. For example, model states can be saved at several intervals for generating ICs for seasonal prediction experiments, or at the end of a long spinup run for starting control/historical experiments on other machines.

Activating the save_ic tool is done by adding save_ic to the **`config`** variable at the top of the runscript. save_ic has two options available:

- save_ic:end_leg (save ICs at the end of every leg)
- save_ic:end_run (save ICs at the end of the run)

By default, without an option, ICs are saved at given offsets from the leg start, possibly if conditions are met. These offset(s) and condition(s) must be coded in the runscript or in the save_ic_get_config() function in runtime/classic/libsave_ic.sh, consult this function for more details. If using the Autosubmit runtime, the SAVE_IC variable is used to activate the save_ic tool (valid values are FALSE, TRUE, end_leg and end_run). SAVE_IC_OFFSET and SAVE_IC_COND control the offset and condition.

Restrictions and issues

The following restrictions apply at time of writing:

- only IFS, NEMO and OASIS (for IFS/NEMO) ICs are generated
- first timestep of run is not supported
- only one IC per month is supported (to simplify the IFS filter)
- maximum 9 ICs per leg (because of current restrictions in NEMO namelist)
- if requesting model-level output for IFS (e.g. for PRIMAVERA), there will be no model-level variables in output on the timestep which ICs are requested, unless requesting the last timestep of the leg

Running EC-Earth 3 Experiments with Autosubmit

Autosubmit is a Python tool to create, manage and monitor experiments in diverse supercomputing environments. It has support for experiments running in more than one supercomputing platform and for different workflow configurations, such as multi-member ensembles including post-processing and archiving tasks. Autosubmit manages the submission of jobs to queue scheduler remotely via ssh, until there is no job left to be run. Additionally, it also provides features to suspend, resume, restart and extend similar experiment at later stage.

Setup prerequisites for using Autosubmit

Autosubmit 3.10 version is available via PyPi package under the terms of GNU General Pub-

lic License. You can find help about how to install and use Autosubmit and a list of available commands in the online documentation:

```
http://www.bsc.es/projects/earthscience/autosubmit
```

Prerequisites: Only a few pre-installed tools/libraries are necessary to use Autosubmit:

- Git, and/or subversion, for repository management
- ssh/scp/rsync, for remote control
- Bash, or any kind of linux shell (on HPC), for script execution
- SQLite, for database management
- Python >=2.7, for script configuration (argparse, dateutil, pyparsing, numpy, pydot-plus, matplotlib and paramiko must be available for Python runtime).

The `Building` section includes instructions about how EC-Earth model can be made available on different HPC.

Thereafter, a multi-member ensemble can be run and monitored with Autosubmit, for example from a laptop (host machine). The host machine has to be able to access HPC via password-less ssh.

Before creating an Autosubmit experiment you have to configure database and path for Autosubmit. It can be done at host, user or local level (by default at host level). If it does not exist, creates a repository for experiments: For example:

```
/home/Earth/autosubmit
```

To create a repository for experiments and database, run the commands:

```
> autosubmit configure
> autosubmit install
```

To create a new experiment, run the command:

```
> autosubmit expid --HPC HPCname --description Description
```

`HPCname` is the name of the main HPC platform for the experiment: it will be the default platform for the tasks. `Description` is a brief experiment description.

This command assigns a unique four character identifier (`xxxx`, names starting from a letter, the other three characters) to the experiment and creates a new folder in the experiments directory tree.

Configuring Autosubmit experiment

An EC-Earth multi-member ensemble climate simulation workflow, due to its length, has to be composed into pieces (or chunks, or legs). Each chunk can be divided into pre-processing, parallel run and post-processing. All these pieces can be submitted, following a given order, to a batch scheduler: we call them jobs.

By default an Autosubmit experiment comes with complete workflow definition. In this guide we modify the default to be simpler. The example 1 (see below) only includes one job for local setup and two jobs of parallel run without pre and post-processing.

EC-Earth 3 includes classic runtime where the legs can be run using the launch function.

EC-Earth 3 includes platform dependent template files and configurable parameters, prepared to run with Autosubmit under the location:

```
ECEARTH3_BASE_DIR/runtime/autosubmit
```

The `Run configuration with ec-conf` section includes instructions about how to prepare the run-scripts to reflect the current platform and the experiment. You can use the same instructions but changing the runtime location to the runtime/autosubmit.

To submit the generated run-scripts with Autosubmit you have to configure the experiment, editing `expdef_xxxx.conf`, `jobs_xxxx.conf` and `platforms_xxxx.conf` in the conf folder of the experiment (see contents in `Illustration 1`).

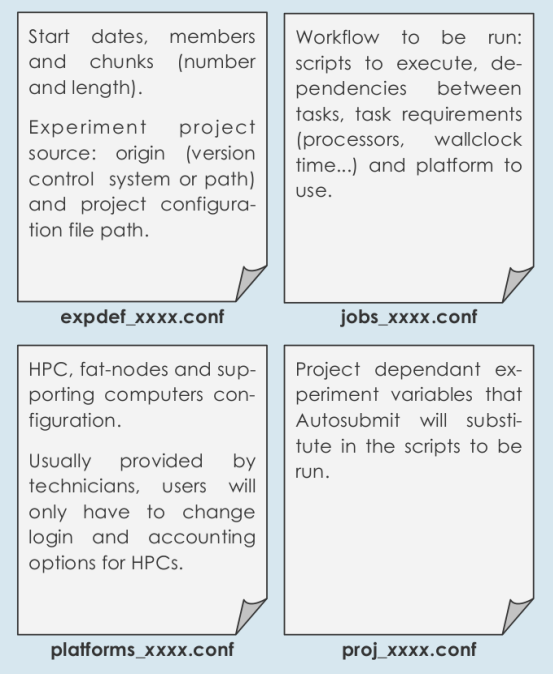


Illustration 1: Configuration files content

Example 1

```
# expdef_xxxx.conf
PROJECT_TYPE = local
PROJECT_PATH = </path/to/ECEARTH3_BASE_DIR>
DATELIST = 19900101
MEMBERS = fc0 fc1


# jobs_xxxx.conf
[LOCAL_SETUP]
```

```
  FILE = copy_runtime.sh
  PLATFORM = LOCAL
  [SIM]
  FILE = ece-ifs+nemo.sh
  DEPENDENCIES = LOCAL_SETUP
  PLATFORM = marenostrum4
  WALLCLOCK = 01:00
  PROCESSORS = 256
```

The Autosubmit experiment creation is launched with the command:

```
> autosubmit create xxxx
```

This command creates the experiment project in the `proj` folder. The experiment project contains the scripts specified in `jobs_xxxx.conf` and a copy of model source code and data specified in `expdef_xxxx.conf`.

Finally, the experiment run is launched with the command:

```
> autosubmit run xxxx
```

This command processes the run-scripts you have specified in the `jobs_xxxx.conf` and substitutes Autosubmit variables such as `START_DATE`, and `MEMBER` and creates `.cmd` files in the `LOG_xxxx` folder that are automatically sent to the specified platform and submitted when the dependencies are fulfilled.

Thereafter, the experiment can be monitored with the command:

```
> autosubmit monitor xxxx
```