# Framework for Ice Sheet - Ocean Coupled modelling (FISOC) Manual, V1.1

Rupert Gladstone and other FISOC contributors (fisoc_dev@googlegroups.com)

June 22, 2020

# Contents

# 1 Introduction

The "Framework for Ice Sheet - Ocean model Coupling" (FISOC) has been written to enable Ice Sheet Models (ISMs) and Ocean Models (OMs) to be run as a single executable to address the co-evolution of ice and ocean properties. It is primarily designed to handle exchange of variables between ice and ocean at the underside of a floating ice shelf. At time of writing (May 2020) exchange of variables at an ice cliff is not supported, though this is a planned development.

In this context an ISM simulates (part of) a marine ice sheet, including both grounded and floating parts, representing the dynamic evolution over time of the ice sheet.

An OM simulates the sub-shelf cavity circulation under the floating part of the ice sheet, and optionally also a wider ocean domain.

FISOC comprises a set of code modules and driver built using the Earth System Modelling Framework (ESMF, `https://www.earthsystemcog.org/projects/esmf/`), and written in Fortran 90. Some knowledge of the ESMF is essential in order to fully understand the FISOC code. It should be possible to run FISOC as an end-user without knowledge of ESMF. FISOC is intended to be flexible: the additional development effort required to couple a different ocean or ice component into the framework is minimal, so long as the new component is ESMF-compatible.

FISOC initially couples the ISM Elmer/Ice to the OM "Regional Ocean Modelling System" (ROMS). A full description of the physical processes represented in FISOC in its initial configuration is given in a model description paper (to be submitted to Geoscientific Model Development, GMD, during 2020).

This manual describes how to use FISOC with an ISM and OM for which it has been developed (Section 3) and also how to integrate additional ISM or OM components into the FISOC framework (Section 5).

A number of web links are given at various points in this document. The links were correct at the time they were added. Please contact the developers if any of these are found to be out of date.

## 1.1 FISOC community and contact

Three email groups exist within the FISOC project.

**fisoc@googlegroups.com** is for news and updates. It has low traffic. Anyone may apply to join this group.

**fisoc_tech@googlegroups.com** is for active users and developers to share technical issues and troubleshooting. It is higher traffic. Anyone may apply to join this group.

**fisoc_dev@googlegroups.com** is the contact address to get in touch with the FISOC developers. Membership is by invitation only, but anyone may post to, and read, this group's contents.

# 2 Installing FISOC with established components

FISOC can be obtained directly from a public github repository:
`https://github.com/RupertGladstone/FISOC`

The most active branch is devel, and master is updated from this infrequently.

FISOC has a simple build process. The Makefile in the top level FISOC directory contains the hard coded dependencies needed to build FISOC code. The Makefile refers to certain environment variables to determine paths and component choices (Section 2.2). An example script to build and install FISOC, "buildFISOCexample.sh", is available in the top level FISOC directory. The script simply sets appropriate environment variables then calls the Make process. The pre-requisites (Section 2.3) must already be installed.

FISOC has been built and used with GNU and with Intel compilers. This has been carried out on different flavours of Linux. Automated testing to validate an install is planned for the future.

## 2.1 Switching between components

Many aspects of FISOC are runtime configurable (Section 3), but the choice of which component to use is a compile time option, implemented by choice of model-specific wrapper. This is determined by environment

| Environment variable | | Description | Default |
|---|---|---|---|
| **ESMFMKFILE** | Required | Tells FISOC where to find ESMF | None |
| **FISOC_INSTALL_DIR** | Optional | Determines where FISOC executable will be installed (should be in $PATH) | $HOME/bin |
| **FISOC_EXE** | Optional | Name of the FISOC executable | FISOC_caller. |
| **CPPFLAGS** | Optional | Preprocessor directives | None |
| **FISOC_ISM** | Required | Determines ISM component ("dummy", "Elmer" or "FOOL") | None |
| **FISOC_ISM_INCLUDE** | Optional | Location of ISM header files | None |
| **FISOC_ISM_LIBPATH** | Optional | Location of ISM library files | None |
| **FISOC_ISM_LIBS** | Optional | Linker directives for ISM library | None |
| **FISOC_ISM_GEOM** | Optional | ESMF geometry object for ISM (FISOC_ISM_GRID or FISOC_ISM_MESH) | FISOC_ISM_MESH |
| **FISOC_OM** | Required | Determines OM component ("dummy", "ROMS" or "FVCOM") | None |
| **FISOC_OM_INCLUDE** | Optional | Location of OM header files | None |
| **FISOC_OM_LIBPATH** | Optional | Location of OM library files | None |
| **FISOC_OM_LIBS** | Optional | Linker directives for OM library | None |
| **FISOC_OM_GEOM** | Optional | ESMF geometry object for OM (FISOC_OM_GRID or FISOC_OM_MESH) | FISOC_OM_GRID |

Table 1: Environment variables that are used in the FISOC build process. Note that **CPPFLAGS** should include **FISOC_MPI** for parallel runs.

variables FISOC_OM and FISOC_ISM (Section 2.2).

In order to test the build process the dummy wrappers may be used. Libraries are not needed for dummy wrappers, and the corresponding environment variables can be set to any value. See "buildFISOCexample.sh" for examples.

In addition to the dummy wrappers, undocumented ISM components include: "FOrcing OffLine" (FOOL), which acts as a wrapper for netcdf files containing time evolving geometry forcing we want to use to force the OM; "Frank's Ice Shelf model" (FISh), a very simple flowline SSA ice shelf model from Frank Pattyn, used for simple testing during development. There is also an undocumented OM component, the Finite Volume Community Ocean Model (FVCOM), which is newly coupled through FISOC and will be documented soon (intended during 2020).

## 2.2 FISOC Environment Variables

A number of environment variables, summarised in Table 1, may be used in the build process. Some of these are mandatory. Variables listed as optional in Table 1 may be mandatory for some configurations other than "dummy". These environment variables are not used at run time, only during the compilation/installation of FISOC.

See also Section 2.3.2 for ROMS-specific preprocessor directives.

## 2.3 Pre-requisites

A Message Passing Interface (MPI) implementation, such as OpenMPI is required.
`http://www.open-mpi.org/`

The Network Common Data Form (NetCDF) Fortran interface must be available. More specifically, NetCDF4 in parallel should be used (this is not the same as PnetCDF).
`http://www.unidata.ucar.edu/software/netcdf/`

ESMF must be available. ESMF should have been built with NetCDF and MPI (see also notes in Appendix A).
`https://www.earthsystemcog.org/projects/esmf/`.

Viable ISM and OM components must be available for any physically meaningful simulations (the build may be tested using "dummy" components). See Sections 2.3.1 and 2.3.2.

### 2.3.1 Elmer/Ice

**Compiling Elmer/Ice for FISOC**
At time of writing (May 2020) FISOC requires a non-standard Elmer/Ice code branch. The Elmer repository is here: `https://github.com/ElmerCSC/elmerfem/`. The standard branch used by glaciologists is the elmerice branch. The branch needed for FISOC is the elmerice_FISOC branch. It is intended to merge the changes during 2020. You can checkout the relevant branch with a command like this:

```
git clone git://www.github.com/ElmerCSC/elmerfem -b elmerice_FISOC MyLocalName
```

There is nothing FISOC-specific about the Elmer/Ice build process. So long as you have the FISOC-compatible code branch, you can build Elmer/Ice just the same as normal. The Elmer/Ice wiki has information on a standard build: `http://elmerice.elmerfem.org/wiki/doku.php?id=compilation:compilationcmake`

**Compiling FISOC with Elmer/Ice**
When compiling FISOC with Elmer/Ice, FISOC needs to know where to find the relevant Elmer/Ice libraries. This can be done at FISOC compile time through the $FISOC_ISM environment variables. For example:

```
export FISOC_ISM="Elmer"
export FISOC_ISM_INCLUDE="$ELMER_HOME/share/elmersolver/include"
export FISOC_ISM_LIBPATH="$ELMER_HOME/lib/"
export FISOC_ISM_LIBS="-lelmersolver"
```

### 2.3.2 ROMS

**Compiling ROMS for FISOC**
FISOC has been developed and tested with an ice shelf enabled version of ROMS. This is branched from the Rutgers ROMS repository. Information about the Rutgers ROMS can be found at `https://www.myroms.org/`.

Development of the ice shelf enabled version is currently ongoing in a private repository (please contact the developers if you need access to this).

Example build scripts can be found in the ROMS/Bin subdirectory of a git clone from the repository mentioned above. It is assumed that users are familiar with a standard ROMS build process.

When compiling ROMS for use with FISOC, the following additional environment variables are needed:

```
export MAKE_SHAREDLIB="Yes"
export LIBDIR="/usr/local/lib"
export MY_CPP_FLAGS=" -DFISOC"
```

It is essential to activate the option to compile the ROMS shared library, which is done by setting the environment variable MAKE_SHAREDLIB to any value. The -fPIC flag is essential, and this will in general be activated by MAKE_SHAREDLIB in a makefile supplement included by a line like this in the makefile:

```
include $(COMPILERS)/$(OS)-$(strip $(FORT)).mk
```

This should simply work, but may require minor modifications on new OS/compiler combinations.

The shared library will be installed in the location given by the LIBDIR environment variable.

The -DFISOC flag activates FISOC-specific code segments. This includes telling ROMS to use a specific hard coded unit rather than outputting to screen. This relies on the same unit being hard coded in the FISOC ROMS wrapper, and results in the ROMS messages being sent to file instead of printed to screen.

Some aspects of the ROMS setup for specific simulations are determined at compile time rather than run time. These are determined through cpp directives set in an application-specific header file in ROMS/Include in the ROMS clone. ROMS must be compiled with an "application" (this is set through an environment variable, see example ROMS build script) that has been setup for use with FISOC. This should give a good indication of which applications are FISOC-compatible:

```
grep ‘‘ifdef FISOC’’ ROMS/Include/*h
```

The cpp options are described in ROMS/Include/cppdefs.h. More detail of the FISOC specific options is given below, but to compile ROMS for use with existing FISOC examples, knowledge of these is not required as the header files will set the required cpp directives and the header file is determined by choice of ROMS application.

To tell ROMS to expect FISOC to provide a cavity change rate:
FISOC_DDDT
To tell ROMS to expect FISOC to provide an upper ice surface change rate (required for grounding line migration and is used to update the wet/dry masks based on floatation):
FISOC_DSDT
To tell ROMS to expect FISOC to directly overwrite the iceshelf draft (this and FISOC_DDDT are mutually exclusive):
FISOC_DRAFT
To tell ROMS to expect to receive a vertical temperature gradient at the ice base from FISOC:
FISOC_DTDZ
To tell ROMS to calculate the averages of the variables provided to FISOC at the end of each ROMS run call:
ROMS_AVERAGES
If ice shelf geometry evolution is required in ROMS this options must be defined:
ICESHELF_MORPH
If grounding line movement is required in ROMS the following cpp options must be defined:
LIMIT_BSTRESS
LIMIT_ICESTRESS
WET_DRY

By default ROMS will install the module files in the directory given by SCRATCH_DIR.

**Compiling FISOC with ROMS**
When compiling FISOC with ROMS, FISOC needs to know where to find the relevant ROMS libraries. This can be done at FISOC compile time through the $FISOC_OM environment variables. For example:

```
export MY_ROMS_DIR="/home/elmeruser/Source/ROMSIceShelf_devel"
export FISOC_OM="ROMS"
export FISOC_OM_LIBS="-loceanM"
export FISOC_OM_INCLUDE="${MY_ROMS_DIR}/Build"
export FISOC_OM_LIBPATH="/usr/local/lib"
```

The ROMS cpp directives require matching cpp directives in the FISOC compilation process. These can be set for a FISOC build using the CPPFLAGS environment variable. For example (see also the example FISOC build script):

```
export CPPFLAGS="$CPPFLAGS -D ROMS_DDDT"
```

At the time of writing, the relevant values are ROMS_MASKING, ROMS_SPHERICAL, ROMS_DDDT, ROMS_DSDT, ROMS_DRAFT and ROMS_AVERAGES. Use of the averaged melt rate from ROMS is activated in FISOC by the ROMS_AVERAGES cpp. The general rule is that any of these directives that are defined for the ROMS compilation need to be defined also for the FISOC compilation.

### 2.3.3 FVCOM

**Compiling FVCOM for FISOC**

FISOC has also been developed and tested with an ice shelf enabled version of unstructured grid Finite Volume Community Ocean Model as described in `https://doi.org/10.1016/j.ocemod.2019.101536`. General information about FVCOM can be found at `http://fvcom.smast.umassd.edu/fvcom/`. Note that this website contains a registration form which must be completed to download the model source code. The module for ice shelf cavity dynamics is available via a static patch (`https://doi.org/10.17632/m6g4c3hm9m.1`) that can be used to augment the standard version of the model, or via the development branch `https://source.coderefinery.org/apn/fvcom4_fisoc` (please contact the authors of the above work for access).

Example build scripts can be found in the directory of a git clone from the repository mentioned above (fvcom4_fisoc). It is assumed that users are familiar with a standard FVCOM build process. buildFVCOM_FX4.sh is the build script for FISOC Example 4 and it calls buildFVCOM_FX4.py that installs required libraries and compiles FVCOM. Any hard coded paths in buildFVCOM_FX4.py should be replaced with corresponding paths to compile FVCOM for use with the Example 4. The build is specific to the machine on which FVCOM/FISOC was developed and tested.

The cpp options for building FVCOM are described in /fvcom4_fisoc/FVCOM_source/make.inc. The FISOC specific options are:

```
CPP_MISC ?= -DICESHELF -DICEDRAFTMORPHY -DFVCOM_API -DFISOC_DDDT -DFISOC_DRAFT
```

Detailed description of the FISOC specific options is given below.

To tell FVCOM to expect FISOC to provide the iceshelf change rate:
FISOC_DRAFT
To tell FVCOM to expect FISOC to provide a cavity change rate:
FISOC_DDDT
To tell FVCOM to expect FISOC to provide an upper ice surface change rate (required for grounding line migration and is used to update the wet/dry masks based on floatation):
FISOC_DSDT
To tell FVCOM to calculate the averages of the variables provided to FISOC at the end of each FVCOM run call:
FVCOM_API
If ice shelf geometry evolution is required in FVCOM this options must be defined:
ICEDRAFTMORPHY
If grounding line movement is required in ROMS the following cpp options must be defined:
WET_DRY

**Compiling FISOC with FVCOM**

When compiling FISOC with FVCOM, FISOC needs to know where to find the relevant FVCOM libraries. This can be done at FISOC compile time through the $FISOC_OM environment variables. For example:

```
export FISOC_OM="FVCOM"
export FISOC_OM_LIBS="-lfvcom_api -lmetis -ljulian"
export FISOC_OM_INCLUDE="${MY_FVCOM_DIR}/FVCOM_source"
export FISOC_OM_LIBPATH="/usr/local/lib"
```

# 3 Running FISOC

The FISOC executable is by default called FISOC_caller, and should be located in your path after installation. The installation is specific to the choice of component (you need to re-compile if you switch, for example, from one OM to another). Beyond choice of components, all run time choices are made in the FISOC_config.rc file (Section 3.1), or through component specific initialisation.

For example, you can run FISOC in serial like this:

```
FISOC_caller
```

You can run FISOC in parallel like this (depending on your system):

```
mpirun -np 4 FISOC_caller
```

In the first instance a dummy coupler can be run by setting both environment variables FISOC_ISM and FISOC_OM to "dummy" at compile time. This can help to test the compilation, and was used during development, but performs no meaningful science.

In verbose mode (Section 3.1) some run time information may be printed to the screen. Independently of this, log files are always written (see Section 3.5).

## 3.1 FISOC runtime configuration

The FISOC configuration file is named FISOC_config.rc, and is expected to be present in the current directory when running FISOC. This is a resource file, as described by the ESMF documentation. It supports different types and also lists. In principle, lists of mixed types are supported, though FISOC does not utilise this capability. Basic syntax highlighting for .rc files can be activated within emacs (see notes in doc/FISOC_emacsMode.asc in the FISOC repository).

FISOC_config.rc mainly contains parameters specific to the coupling rather than to the running of individual components. Components should use their standard means for configuration, and component-specific configuration files should be specified in the FISOC configuration file.

Some of the FISOC config entries are strictly required and some are optional. This section describes all the valid standard config entries. Note that model-specific non-standard entries can be added if needed, and use of these should be through the model-specific wrapper code.

**ISM options**

**ISM_configFile:** [STRING] [optional]
The name of the ISM-specific config file.

**ISM_stdoutFile:** [STRING] [optional]
The name of a file to which to write the ISM standard output. May be required depending on model-specific wrapper.

**FISOC_ISM_ReqVars:** [STRING] [required]
List of variable names required to be provided by the ISM.

**ISM_varNames:** [STRING] [optional]
List of native names of variables in the ISM, for use by the model-specific wrapper. May also be required or unused depending on the wrapper. Must be same length as **FISOC_ISM_ReqVars**.

**FISOC_ISM_DerVars:** [STRING] [required]
List of variables derived by FISOC from the ISM variables. These are calculated from ISM required variables by hard coded routines in FISOC_ISM or FISOC_utils. This list is allowed to be empty.

**ISM_maskOMvars:** [LOGICAL] [optional]
Determines whether the ISM_maskOMvars should be turned on or not. The generic pre-requisite for this is that ISM_gmask must be present, i.e. it must be a member of either FISOC_ISM_ReqVars or FISOC_ISM_DerVars. The Elmer-specific pre-requisite is that the Elmer variable GroundedMask must exist according to the .sif.

**ISM2OM_vars:** [STRING] [optional]
List of variables to be passed from the ISM to the OM. Defaults to all ISM variables (union of required and derived variables). If an empty list is given no variables will be passed from the ISM to the OM. This can be useful for unit testing.

**ISM2OM_init_vars:** [LOGICAL] [optional]
Determines whether the ISM2OM_vars should be passed to the OM during the second phase of OM initialisation. Default is .TRUE.

**ISM2OM_regrid:** [STRING] [optional]
The ESM regridding method to use. If not set, the default will be written to the log files. Possible values are given in the ESMF documentation. `http://www.earthsystemmodeling.org/esmf_releases/public/last/ESMF_refdoc/node9.html#SECTION090146000000000000000`

**ISM2OM_extrap:** [STRING] [optional]
Some ESMF regridding methods may be supplemented with an extrapolation method for target points outside the source domain.

## OM options

**OM_configFile:** [STRING] [optional]
The name of the ISM-specific config file.

**OM_stdoutFile:** [STRING] [optional]
The name of a file to which to write the OM standard output. May be required depending on model-specific wrapper.

**FISOC_OM_ReqVars:** [STRING] [required]
List of variable names required to be provided by the OM.

**OM_ReqVars_stagger:** [STRING] [optional]
Corresponding exactly to **FISOC_OM_ReqVars**, descriptions of the grid stagger for each variable.

**FISOC_OM_DerVars:** [STRING] [required]
List of variables derived by FISOC from the OM vars. To be calculated from OM vars by hard coded routines in FISOC_OM.

**OM2ISM_vars:** [STRING] [optional]
List of variables to be passed from the OM to the ISM. Defaults to all OM variables (union of required and derived variables). If an empty list is given no variables will be passed from the OM to the ISM. This can be useful for unit testing.

**OM2ISM_init_vars:** [LOGICAL] [optional]
Determines whether the OM2ISM_vars should be passed to the ISM during the second phase of ISM initialisation. Default is .TRUE.

**OM_initCavityFromISM:** [LOGICAL] [optional]
Switch to allow the OM to overwrite its cavity geometry with ISM_z_l0 during the second phase of initialisation. Defaults to .FALSE.

**OM_cavityUpdate:** [STRING] [optional]
How to process ISM ice draft for use in OM. Valid values are RecentIce
(default), Rate, CorrectedRate, and Linterp.

**OM_WCmin:** [REAL] [optional]
Minimum water column thickness imposed by OM. Defaults to zero. When using ROMS, this corresponds to the ROMS DCRIT (in the .in file) and should be set to the same value. Only used with CorrectedRate to preserve a "dry" water column under grounded ice.

**OM_CavCorr:** [REAL] [optional]
The proportion of the OM - ISM cavity discrepancy to correct in one FISOC call to the OM run method. This cavity correction factor can take values from 0 to 1. Defaults to 0.2. Only used with CorrectedRate.

**OM_outputInterval:** [INTEGER][optional]

FISOC collects OM output once every OM_outputInterval OM timesteps. Defaults to 1. dt_ratio/OM_outputInterval must be integer.

**OM2ISM_regrid:** [STRING] [optional]
The ESMF regridding method to use. If not set, the default will be written to the log files. Possible values are given in the ESMF documentation. `http://www.earthsystemmodeling.org/esmf_releases/public/last/ESMF_refdoc/node9.html#SECTION090146000000000000000`

**OM2ISM_extrap:** [STRING] [optional]
Some ESMF regridding methods may be supplemented with an extrapolation method for target points outside the source domain.

**Netcdf output options**
Note: these are considered OM options because the output writing occurs from the OM generic wrapper. The OM import state here contains the ISM variables regridded to the OM grid.

**OM_writeNetcdf:** [LOGICAL] [optional]
Switch for dumping the OM import and export variables to NetCDF files. Defaults to .TRUE.

**output_dir:** [STRING] [optional]
Path to directory (must already exist) to which to write the NetCDF files. Defaults to current directory.

**OM_NCfreq:** [STRING] [optional]
Output writing frequency. Defaults to "all". Valid values are "all" or "ISM" (only write netcdf outputs after an ISM timestep).

**Timestepping options**
**OM_dt_sec:** [INTEGER][required]
OM timestep length in seconds. When using ROMS, this may be a multiple of the ROMS timestep length, in which case each FISOC call to the ROMS run method will run multiple ROMS timesteps.

**dt_ratio:** [INTEGER][required]
ISM/OM timestep ratio.

**start_year:** [INTEGER][required]
Start year and month define the start time of the coupled simulation.

**start_month:** [INTEGER][required]

**end_year:** [INTEGER][optional]
End year and month define the finish time of the coupled simulation.

**end_month:** [INTEGER][optional]

**runLength_ISM_steps:** [INTEGER][optional]
As an alternative to specifying an end time, the run length in terms of ISM timesteps may be specified.

**General options**
**verbose_coupling:** [LOGICAL][required]
If true, some run time information is printed to the screen. A log file is always written, but writing to the log during timestepping is suppressed when verbose_coupling is false.

### 3.1.1 FISOC variables

The union of **FISOC_ISM_ReqVars**, **FISOC_ISM_DerVars**, **FISOC_OM_ReqVars** and **FISOC_OM_DerVars** describes the full set of variables required by FISOC for a given simulation. Valid values are given in Table 2.

Note that the units given in Table 2 are suggested units. FISOC doesn't care about units, but the user must ensure unit consistency. There may be hard coded unit assumptions in the model specific wrappers.

As a naming convention, "z" refers to the vertical coordinate, and "l0" and "l1" refer to the model level at the ice-ocean interface (this would typically be the lowest level of the ISM or the uppermost level of the OM) and one level above it, respectively. "lts" refers to the top surface of the ISM.

FISOC outputting occurs on the ocean grid, and consists of dumping both the import and export fields to netcdf files. **FISOC_ISM_ReqVars** may contain variables that are required only so that they can be written out on the ocean grid (typically as a sanity or regridding check) rather than actually being needed by the OM.

The list of derived variables, **FISOC_ISM_DerVars**, indicates which variables are needed by the OM but are not calculated by the ISM or its wrapper. The methods for calculating the derived variables are hard coded in FISOC_ISM.f90. Valid values for **FISOC_ISM_DerVars** include:

ISM_z_l0_previous. This is the depth of ice base at previous ISM timestep. This is simply stored in memory. No calculation is required, but this variable is needed for the other "derived" variables.

ISM_dTdz_l0. Temperature gradient at ice base. This is calculated as

$$ISM\_dTdz\_l0 = \frac{ISM\_temperature\_l1 - ISM\_temperature\_l0}{ISM\_z\_l1 - ISM\_z\_l0} \tag{1}$$

ISM_dddt. Rate of change of depth of ice base. This is calculated as

$$ISM\_dddt = \frac{ISM\_z\_l0 - ISM\_z\_l0\_previous}{ISM\_dt} \tag{2}$$

ISM_dsdt is calculated similarly to ISM_dddt.

Not all ISM variables need to be passed to the OM, and vice versa. This choice is made by the user through use of configuration options ISM2OM_vars and OM2ISM_vars. These options alow the user to specify a subset of the full set of required and derived variables that will be passed to the other component in a given simulation. An empty list can be used to avoid any variables being passed between components. This can be useful during testing and troubleshooting.

A note on efficiency (May 2020): All the ISM and OM variables are currently being regridded and passed to the wrapper for the opposite component. It is at the model-specific wrapper level that ISM2OM_vars and OM2ISM_vars are checked. If regridding becomes a significant proportion of FISOC's computational cost this should be re-implemented to reduce non-essential regridding operations.

### 3.1.2 Updating the ice shelf cavity for the OM

Several options are available through FISOC for updating the OM representation of the ocean cavity. These vary from the simplest option of using the most recent cavity geometry from the ISM to update the OM representation of cavity geometry to smoother options via either interpolation in time or specifying a rate of change of cavity geometry.

The options (summarised in Table 3) are all implemented within FISOC, based on the cavity geometry calculated by the ISM. Some options are through FISOC derived variables, as described in Section 3.1.1. The key ISM output from which all cavity options are calculated is ISM_z_l0.

The FISOC GMD paper will provide more details about these approaches to updating the ocean representation of the ice shelf cavity, and also about the wetting and drying scheme used for grounding line migration.

## 3.2 Timestepping

Different asynchronous timestepping options are planned. Currently recommended (May 2020) is to set dt_ratio = 1 and use the ISM timestep for both the ISM and OM components. ROMS or FVCOM will run as many timesteps as needed (see also DT in the ROMS .in file) for each call to their run method. With this approach the Rate or CorrectedRate methods for cavity evolution are recommended. FISOC can pass the ROMS melt rate to the ISM after the call to the ROMS run method.

| Variable | Description | Units |
|---|---|---|
| ISM_temperature_l0 | Ice temperature at the ice ocean interface. | K |
| ISM_temperature_l1 | Ice temperature in the ISM one level above the ice ocean interface. | K |
| ISM_z_l0 | Height relative to sea level of the ice base. | m |
| ISM_z_lts | Height relative to sea level of the ice top surface. | m |
| ISM_z_l1 | Height relative to sea level of the first ISM model level above the ice base. | m |
| ISM_z_l0_previous | Height relative to sea level of the ice base one ISM timestep previously. | m |
| ISM_z_lts_previous | Height relative to sea level of the ice top surface one ISM timestep previously. | m |
| ISM_thick | Ice thickness | m |
| ISM_dTdz_l0 | Vertical temperature gradient in the ice at the ice base. | K/m |
| ISM_dddt | Rate of change of the ice draft with respect to time. | m/a |
| ISM_dsdt | Rate of change of the ice top surface height with respect to time. | m/a |
| ISM_velocity_l0 | Ice flow velocity at the ice base | m/a |
| ISM_maskOMvars | Mask the fields being passed from OM to the ISM | |
| OM_bmb | Ice shelf basal melt rate. | m/a |
| OM_temperature_l0 | Ocean temperature at the ice ocean interface. | |
| OM_z_l0 | Height relative to sea level of the ice base. | m |
| OM_bed | Ocean bathymetry | |
| OM_z_lts | Height relative to sea level of the ice top surface. | m |

Table 2: FISOC standard variables and typical units. Note that heights relative to sea level are always positive upward. Some variables in this table differ only in their prefix (ISM or OM). The only differnce is that they are defined on the geometry object (grid or mesh) of their respective component. For example ISM_z_lts is needed to pass the ice upper surface from the ISM to the OM, and OM_z_lts is needed in order to derive a drift correction if the correctedRate cavity option is used.

| Cavity option | Summary | Required ISM2OM variable |
|---|---|---|
| RecentIce | Most recent ice draft from ISM | ISM_z_l0 |
| Rate | Rate of change of ice draft from two most recent ISM steps | ISM_dddt |
| CorrectedRate | As above with additional drift correction | ISM_dddt |
| Linterp | Time-interpolated ice draft from two most recent ISM steps | ISM_z_l0_linterp |

Table 3: Cavity update options. Note that of the possible ISM cavity variables (ISM_z_l0, ISM_z_l0_linterp, ISM_dddt) only the required variable (third column) should be passed to the OM (constrained using ISM2OM_vars). At time of writing (May 2020) the Rate and CorrectedRate are giving the most reliable outcomes and are recommended.

## 3.3 Running FISOC with Elmer/Ice

For dynamic linked libraries, shared object files may be needed at run time. This can be ensured through use of the $LD_LIBRARY_PATH environment variable.

For example (it is assumed $ELMER_HOME was set during Elmer installation):

```
export LD_LIBRARY_PATH="$FISOC_ISM_LIBPATH/:$LD_LIBRARY_PATH"
```

As with a normal Elmer/Ice simulation, the mesh should be partitioned into the same number of partitions as the number of processors (which is the same as the number of ESMF PETs and DEs).

More information about Elmer, and especially Elmer/Ice, can be found on several sources on the internet.

```
https://www.csc.fi/web/elmer
http://www.nic.funet.fi/pub/sci/physics/elmer/doc/
http://elmerice.elmerfem.org/
http://elmerice.elmerfem.org/wiki/doku.php
http://www.elmerfem.org/forum/
```

### 3.3.1 Elmer/Ice specific configuration

The following options in the FISOC configuration file are used by Elmer/Ice. These are non-standard configuration options.

**ISM_BodyID:** [INTEGER] [required]
The body ID of the surface on which interactions with the ocean occurs. Typically this will be the lower surface, defined as a boundary in the Elmer/Ice mesh file and as a body in the boundary condition section of the .sif.

## 3.4 Running FISOC with ROMS

FISOC needs to access the shared library at run time. One way of ensuring this is to add the location of the library to the $ LD_LIBRARY_PATH variable, e.g.:

```
export LD_LIBRARY_PATH="$FISOC_OM_LIBPATH/:$LD_LIBRARY_PATH"
```

ROMS writes a lot of information to the screen when run alone. When run through FISOC this is redirected to a text file. The file name is given by **OM_stdoutFile**, which must be provided in the FISOC config file whenever ROMS is used.

The number of processes to launch FISOC with must be consistent with the number of partitions in the ROMS grid. This is set in the **OM_configFile** (i.e. the ROMS .in file) by the Ntile parameters. For example, the following gives 4 partitions

```
NtileI == 1                          ! I-direction partition
NtileJ == 4                          ! J-direction partition
```

Some of the ROMS configuration information is in a .dat file. When running ROMS through FISOC, the name and full path of this file must be given to VARNAME in the ROMS .in file.

### 3.5 FISOC output files

The text file outputs from FISOC comprise:
1. Standard out from the job. In interactive runs it is written to the screen.
2. Standard out from the ice model. This is redirected to a file named in the FISOC_config file.
3. Standard out from the ocean model. This is redirected to a file named in the FISOC_config file.
4. FISOC/ESMF log files (usually named PET*.log). These provide line numbers on which errors occurred.

FISOC can output basic netcdf files from the OM generic wrapper on the OM grid. FISOC can do this for both ISM and OM variables. This capability is mainly recommended just for sanity checking.

FISOC/ESMF log files have default filenames of "PET#.FISOC.Log", where "#" is a number from 0 upwards indicating the "Persistent Execution Thread" (PET). These logs are created by FISOC/ESMF and contain run time messages and errors that can be helpful with troubleshooting. Note that by default FISOC appends to the logs rather than over-writing at run time, so you may wish to delete old logs periodically.

Aside: PET is an ESMF abstraction designed to be general over differing parallel implementations. In FISOC, there is always a 1:1 relationship between PETs and MPI processes.

### 3.6 Troubleshooting

If the error messages to the screen are not helpful, remember to check whether useful information has been logged in any of the text file outputs (Section 3.5). In particular, the ESMF/FISOC log files contain line numbers in the code for all logged entries, including errors. These can be used to identify which code segment caused errors. FISOC aims to provide sufficient troubleshooting information in these files. However, not all problems are neatly reported at time of writing (May 2020). Some examples of other errors are given below.

A segmentation fault has been known to occurr in the case of an incorrect path to the ROMS configuration file (the .in file).

Note that DMUMPS error codes, should they occur, can be found in the MUMPS user guide. MUMPS is often used by Elmer/Ice. `http://mumps.enseeiht.fr/`

Errors like the following can ocurr (in the log files) when the number of processes is not consistent with the number of component partitons (this example involves ROMS):

```
20151119 112603.491 ERROR        PET0 ESMCI_DistGrid.C:1200 ESMCI::DistGrid::create() Value
    ↪ unrecognized or out of range - deBlockList contains out-of-bounds elements
20151119 112603.491 ERROR        PET0 ESMCI_DistGrid_F.C:152 c_esmc_distgridcreatedb() Value
    ↪ unrecognized or out of range Internal subroutine call returned Error
20151119 112603.491 ERROR        PET0 ESMF_DistGrid.F90:1220 ESMF_DistGridCreateDB() Value
    ↪ unrecognized or out of range - Internal subroutine call returned Error
20151119 112603.491 ERROR        PET0 src/FISOC_OM_Wrapper_ROMS.f90:612 Value unrecognized or out of
    ↪  range - Passing error in return code
20151119 112603.491 ERROR        PET0 src/FISOC_OM_Wrapper_ROMS.f90:120 Value unrecognized or out of
    ↪  range - Passing error in return code
```

Errors like the following can occur if a component wrapper attempts to access a field that has not been created by FISOC, i.e. a field that is not in the list of required variables in the FISOC config file (Section 3.1) (this example involves FISh):

```
20151207 152613.483 ERROR        PET0 ESMCI_Container_F.C:165 ESMCI::Container::get() Invalid
    ↪ argument key does not exist
20151207 152613.484 ERROR        PET0 ESMCI_Container_F.C:448 c_esmc_containergetfield() Invalid
    ↪ argument Internal subroutine call returned Error
```

```
20151207 152613.484 ERROR        PET0 ESMF_Container.F90:589 ESMF_ContainerGetField() Invalid
    ↪ argument - Internal subroutine call returned Error
20151207 152613.484 ERROR        PET0 ESMF_FieldBundle.F90:1456 ESMF_FieldBundleGetItem() Invalid
    ↪ argument - Internal subroutine call returned Error
20151207 152613.484 ERROR        PET0 src/FISOC_ISM_Wrapper_FISh.f90:198 Invalid argument - Passing
    ↪ error in return code
```

## 3.7 FISOC examples

Example FISOC configurations can be found in the examples subdirectory of the repository.

The ROMS setup for these examples is defined in the ROMS repository, not in the FISOC repository. This is because some aspects of the ROMS setup are defined at compile time and it is standard ROMS development practice to use the ROMS repository for such details. See below for specifics.

The Elmer/Ice setup is defined in the FISOC repository. More information about running the examples can be found in examples/README.

In general it will be necessary to recompile ROMS and FISOC when switching between different examples, but it will not be necessary to recompile Elmer/Ice.

Examples 4 and 5 are described further in the FISOC GMD paper, where they are referred to as Verification Experiments 1 and 2 (VE1 and VE2), along with presentation of outputs.

### 3.7.1 Example 1: Long thin marine ice sheet

A FISOC example using Elmer/Ice and ROMS. At time of writing (7/12/2017) this has not been recently used, may not work, and may be superceded by example 5. The Elmer/Ice setup is provided with the example in the subdirectory. The corresponding ROMS header file is ROMS/Include/iceshelf2d.h, the application is ICESHELF2D, and the input file is ROMS/External/ocean_iceshelf2d.in.

### 3.7.2 Example 2: Ice cliff

Placeholder! To be developed...

### 3.7.3 Example 3: Using FISOC to handle time evolving forcing

A FISOC example using offline forcing to drive ROMS. This is used for running the ISOMIP+ ocean 3 and 4 experiments. The corresponding ROMS header file is ROMS/Include/isomip_plus.h, the application is ISOMIP_PLUS, and the input file is ROMS/External/ocean_isomip_plus_ocn3.in. Processed netcdf files based on those available through MISOMIP are also required. The netcdf files are not included in the FISOC repository.

### 3.7.4 Example 4: Simple floating only test

A FISOC example using Elmer/Ice and ROMS. No grounding line is included. The Elmer/Ice setup is provided with the example in the subdirectory. The corresponding ROMS header file is ROMS/Include/iceshelf2d_toy.h, the application is ICESHELF2D_TOY, and the input file is ROMS/External/ocean_iceshelf2d_toy.in. This example is used in the FISOC GMD model description paper where it is referred to as Verification Experiment 1 (VE1_ER).

### 3.7.5 Example 5: Simple grounding line migration test

A FISOC example using Elmer/Ice and ROMS. Similar domain to example 4, but with an evolving grounding line. The Elmer/Ice setup is provided with the example in the subdirectory. The corresponding ROMS header file is ROMS/Include/iceshelf2d_toy_gl, the application is ICESHELF2D_TOY_GL, and the input file is ROMS/External/ocean_iceshelf2d_toy_gl.in. This example is used in the FISOC GMD model description paper where it is referred to as Verification Experiment 2 (VE2_ER).

| Script name | Dependencies | Function |
|---|---|---|
| griddata_fvcom.m | Matlab | Regrid FVCOM output to regular grid. |
| q_read_fvcom_var.m | Matlab, griddata_fvcom.m | Wrapper for griddata_fvcom.m. |
| ROMSvolume.py | Python, Netcdf4 module | Calculate ROMS total ocean volume. |
| plotVols.m | Matlab, ROMSvolume.py | Line plots of Elmer, ROMS and FVCOM volume or mass. |
| read_from_roms.m | Matlab | Read ROMS gridded data. |
| velPlots.m | Matlab, read_from_roms.m, q_read_fvcom_var.m | Comparative ocean velocity plot. |
| ROMS2Para.m | Matlab | Prepare ROMS netcdf file for Paraview. |
| ElmerGroundedArea.py | Python, Paraview module | Calculate Elmer/Ice total grounded area over time. |
| IntegrateMelt.py | Python, Paraview module | Integrated Elmer/Ice total basal melt over time. |
| ROMSgroundedArea.py | Python, Paraview module | Calculate ROMS total dry cell area over time. |

Table 4: Summary of example output processing scripts. These are not intended to be robust and are not documented. They are provided only as examples.

# 4 Post processing

OM and ISM components should be able to provide their usual output formats, and the default expectation is that standard approaches to visualising and processing outputs will be used for OM and ISM components separately.

The FISOC repository provides some limited functionality for output processing and visualisation. The scripts provided are not robust, and are provided more as examples than as a post processing or visualisation framework. These processing scripts are not supported by FISOC developers. A future intention is to provide more robust and documented post processing utilities.

Elmer/Ice provides .vtu files that are viewable in Paraview. ROMS provides netcdf files that can be made viewable in Paraview with a small amount of manipulation. The ROMS outputs need to be provided at rho points, and the vertical coords need to be processed to express variables on depth levels instead of sigma coordinates. This way it is possible to view both ROMS and Elmer/Ice outputs together interactively through Paraview. This approach uses Netcdf Operators (NCO) and Matlab.

ROMS Matlab repository information is available here: https://www.myroms.org/wiki/Matlab_Scripts. You will need a ROMS account to access the matlab scripts in their subversion repository. In particular set_depth.m is needed if using the ROMS2para.m function provided here (and set_depth calls another ROMS matlab function...).

An example ncrcat command for processing ROMS output files is given here. This command concatenates a subset of the ROMS output data into one netcdf file. For example 5:

```
ncrcat -O -p . -d ocean_time,,,2 -n 72,2,1 ocean_his_0001.nc ocean_his_select.nc -v ocean_time,
    ↪ x_rho,y_rho,Sb,Tb,draft,zeta,m,ubar_eastward,vbar_northward,w,u_eastward,v_northward,temp,
    ↪ salt,h,wetdry_mask_rho,Vtransform,Vstretching,theta_s,theta_b,hc
```

The concatenated file should be suitable for processing through the matlab scripts that prepare for reading ROMS netcdf files into Paraview.

The ROMS2Para Matlab function is provided in the FISOC_pp directory of the FISOC repository. This works for FISOC example 5, but may not work for other ROMS domains depending on the coordinate system.

There are also some example scripts using either Matlab or the Paraview simple module or the Netcdf4 module for Python to perform certain specific postprocessing tasks. These are not documented but are summarised

| **Generic module** | **Role** |
|---|---|
| FISOC_caller.f90 | The FISOC calling program |
| FISOC_parent.f90 | FISOC's top level ESMF gridded component |
| FISOC_coupler.f90 | ESMF coupler component, handles regridding |
| FISOC_ISM.f90 | ESMF gridded component, top level control for ISM |
| FISOC_OM.f90 | ESMF gridded component, top level control for OM |
| FISOC_Utils.f90 | Assorted FISOC utilities, available to all modules |
| | |
| **Component-specific module** | |
| FISOC_OM_Wrapper_XXX.f90 | Model-specific wrapper for OM XXX |
| FISOC_ISM_Wrapper_XXX.f90 | Model-specific wrapper for ISM XXX |

Table 5: Summary of the role of key FISOC code modules. ISM is short for Ice Sheet Model and OM is short for Ocean Model.

in Table 4.

# 5 FISOC design and development

This section describes aspects of FISOC design/development that an end user would typically not need to know. An overview of FISOC code modules is summarised in Table 5 and this information is presented visually in Figure 1. FISOC is intended to be flexible from top down and bottom up: FISOC could be called as part of a larger ESMF coupled model by interfacing to FISOC_parent as a shared library (as of May 2020 the code is not correctly distributed between caller and parent; please contact the developers if you wish to be able to call FISOC as an ESMF component and we can address this issue). FISOC calls independent ice or ocean components through model specific wrappers.

The FISOC_ISM and FISOC_OM modules provide top level control and processing for ice and ocean components. These are not model-specific. The model specific code, which exchanges fields between ESMF structures and structures used by the individual models, is in the FISOC_(O/IS)M_Wrapper modules. The main ISM or OM call structure must be made to be compatible with ESMF, which essentially means having seperate initialise, run and finalise calls. See the ESMF documentation.

## 5.1 ESMF run time objects

FISOC maps component grids or meshes to ESMF_grid or ESMF_mesh objects within the model specific wrappers. FISOC also maps component "variables" or "fields" to ESMF_field objects in the model specific wrappers. These objects contain pointers to the ESMF_grid or ESMF_mesh objects. Multiple ESMF_field objects are wrapped in ESMF_fieldbundle objects.

ESMF_state objects store all data for a gridded component. The model-specific wrappers do not use this level of ESMF code abstraction. Instead the field bundles are packed/unpacked to/from state objects within the non model-specific wrappers (i.e. FISOC_OM and FISOC_ISM modules).

The ESMF_routeHandle objects are used to store weights and all information needed for regridding operations. The coupler component within FISOC sets these up during initialisation using ESMF_mesh or ESMF_grid objects obtained from the OM and ISM components.

The ESMF virtual machine (ESMF_VM) object provides a generic parallel context and can contain information for multiple possible parallelism paradigms. ESMF persistent execution threads (PETs) are also generic objects, representing individual threads within the parallel context. FISOC does not aim for this level of generality. FISOC is intended to use only the message passing interface (MPI), and requires a one to one mapping between MPI processes, PETs and domain partitions within geometry objects. FISOC uses ESMF methods to intialise the parallel context. FISOC then extracts an MPI communicator from the ESMF_VM object and passes this to the ISM and OM components to use in their initialisation phases.

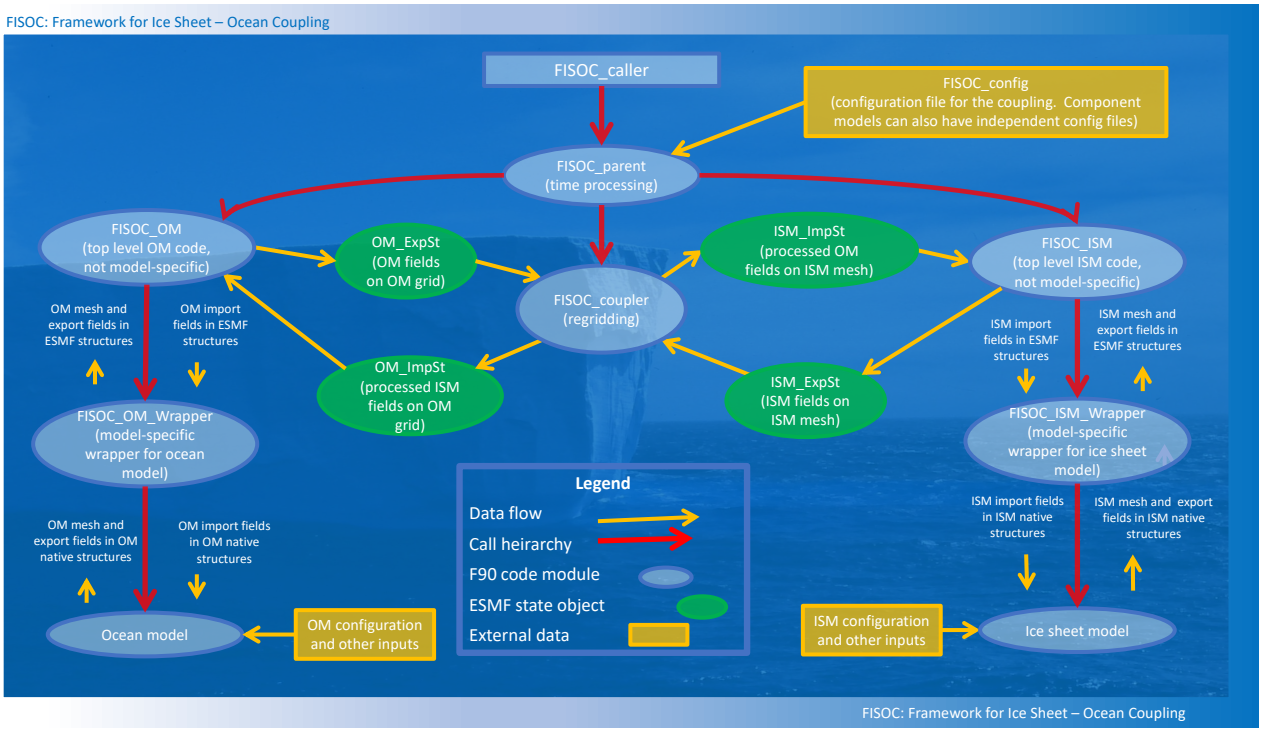


Figure 1: FISOC code and data structures.

FISOC also uses EMSF objects for time keeping, but these are not discussed here.

## 5.2 Incorporating new OM or ISM components

Structural changes to FISOC would be needed to introduce a new type of component, e.g. an atmosphere model. Implementing an alternative ISM or OM should require no changes to existing FISOC code, just an additional model-specific wrapper.

Any new OM or ISM component to be used with FISOC must first be ESMF compliant. This basically means that it should have an initialise, run and finalise routine, and that the developer can provide the component's grid and variables in ESMF compatible structures at run time through the new wrapper. The ESMF web site provides further documentation. `https://www.earthsystemcog.org/projects/esmf/`

The new wrapper must contain a Fortran module with restrictions on the module name and on the interface for the initialise, run and finalise methods. The precise requirements are not documented here as they may evolve during continued development. It is recommended to view an existing wrapper, especially the PUBLIC routines.

If it is found that changes to other aspects of the FISOC code are required, this should be implemented in collaboration with the FISOC core developers.

## 5.3 Coding practices

A new component wrapper should be in a Fortran 90 module. All modules should contain the "implicit none" statement at the top (immediately after any "USE" statements). This property will be inherited by all procedures in the module.

FISOC modules have the private attribute, with only required procedures being made public. New model-specific wrapper modules should ensure that the initialise, run and finalise subroutines are public.

Please implement your developments in a new branch in the FISOC repository. Request a merge to master when you are happy that it is stable.

Adding new ISM or OM components should not require changes to the Makefile. If you feel that changes to the Makefile are needed, please first contact the developers to discuss whether this can be achieved through a build script.

### 5.3.1 Error handling

ESMF provides defensive error handling, with error codes and error messages passed up the call stack. FISOC implements "fail-fast" error handling, with errors generally being considered fatal. Exceptions to this may be made where it is safe to do so (e.g. where a default value can be used in the event of failure to find a config parameter). All calls to ESMF routines have return codes checked immediately, and errors logged. If components (OM or ISM) provide a return code or error code, this should be checked by the component wrapper.

Note that many FISOC subroutines contain a return code, but these are mostly not used in practice. Since errors are generally considered instantly fatal in FISOC, execution will generally not get as far as returning a failure code. If a more defensive rather than fail-fast approach is adopted in the future these return codes can be used.

## 5.4 Configuration options

Compile time choices include which ISM or OM component to run, the type of ESMF geometry object for each component (mesh or grid), and paths to find component libraries. These are set through environment variables during the build process. Other options are mostly run time options, determined by parameters in the FISOC configuration file.

The configuration file must be named "FISOC_config.rc". It is compatible with ESMF config methods. An ESMF_config object is automatically created from this file at run time. This object is always named "FISOC_config" in the FISOC code. New model-specific parameters needed by the new wrappers may be introduced to the config file and can be accessed within the model-specific wrapper at run time, via "FISOC_config". No further coding is needed for this functionality. See existing model-specific wrappers (e.g. FISOC_ISM_Wrapper_Elmer.f90) for examples of this.

The master list of standard FISOC configuration options is contained in FISOC/doc/FISOC_emacsMode.asc and in Section 3.1 of this manual. When new standard configuration options are added during development, they should also be added to both the emacsMode file and Section 3.1 of the manual.

### 5.4.1 Default values and derived attributes

The config file is intended to avoid duplication of information. In the case of configuration options that can be derived from other configuration options, their derivation is hard coded into FISOC_utils.f90 (see FISOC_ConfigDerivedAttribute interface) rather than adding redundant parameters to the config file. It is recommended (though not strictly required) that further developments also follow this approach.

Note that the approach of derived attributes can be used to hard code a default value for an attribute that is in essence not a derived attribute. Using the derived attribute subroutines as a wrapper in which to hard code a default value has the benefit of entering the hard coded value only at one location rather than each time the attribute is accessed from the config object. It has the disadvantage that the utils module is not the most intuitive place to store hard coded defaults. Where defaults are set by the utils module, an info statement should be written to the PET logs.

The derived attributes and default values are not fully documented here due to the dynamic nature of the code base; the code base should be considered to contain the definitive list. Some indication can be obtained by searching the code. This command, for example, shows the lines surrounding where FISOC catches a "not found" error from ESMF:

```
grep -i -C 3 ESMF_RC_NOT_FOUND FISOC_utils.f90
```

Note that it is not intended for physical parameters to have hard coded defaults given using this mechanism. It is dangerous to provide values for physical parameters in more than one location. Use of derived attributes to set defaults is aimed at safe parameters e.g. output frequency.

### 5.5 Sequential parallelism

ESMF supports sequential and concurrent parallel coupling, and any combination of these. FISOC currently (May 2020) supports only sequential coupling, and requries that the ISM and OM alternate use of the same set of processors. This is implemented by duplicating the MPI context from the ESMF virtual machine (VM) and passing this to each component. This can be made more flexible in the future if needed.

The Table to the right summarises the order of events within this sequential coupling paradigm. This ordering is hard coded, but again, this could be made more flexible in the future if needed. Note that there are two initialisation phases so that the ISM and OM components have the chance to modify their initial state based on the initial state of the other component (e.g. the OM may wish to update its ice shelf cavity geometry directly from the initialised ISM during OM intialisation phase 2). Note that the run phases are repeated as many times as are required to complete the simulation. The coupler component has multiple phases in order to separately handle the regridding in each direction (ice to ocean and ocean to ice). ESMF places no limitation on the number of initialisation or run phases for each component, although FISOC currently imposes this fixed ordering, hard coded into FISOC_parent.

| **FISOC component event order** |
|---|
| OM intialisation phase 1 |
| ISM intialisation phase 1 |
| Coupler intialisation phase 1 |
| OM intialisation phase 2 |
| Coupler intialisation phase 2 |
| ISM intialisation phase 2 |
| OM run |
| Coupler run phase 1 |
| ISM run |
| Coupler run phase 2 |
| OM finalise |
| ISM finalise |
| Coupler finalise |

# A Pre-requisite installation notes

The following commands worked to install NetCDF and ESMF on a Linux Mint system in 2015 in a suitable configuration for use with FISOC. Some pre-requisites for netcdf were also installed. The system already had a working OpenMPI installation. The following is just an example, it is not intended as a usable script.

```
# instructions on installing ESMF can be found here:
# http://www.earthsystemmodeling.org/esmf_releases/last_built/ESMF_usrdoc/node9.html

# netcdf instructions
# http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-install.html

cd /somewhere/to/download/and/compile/source/code

sudo apt-get install m4

wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4/zlib-1.2.8.tar.gz
tar -xzf zlib-1.2.8.tar.gz
cd zlib-1.2.8
 ./configure --prefix=/usr/local/
 make check
 sudo -E make install
cd ..

wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4/hdf5-1.8.13.tar.gz
tar -xzf hdf5-1.8.13.tar.gz
cd hdf5-1.8.13
 # Note the O0 flag in the next line. The default is O3,
 # which is strong optimisation. This can result in failed
 # checks on some systems.
 CFLAGS="-O0 -fPIC" CC=mpicc CXX=mpiCC FC=mpif90 ./configure --prefix=/usr/local/ --with-zlib=/usr/
     ↪ local --enable-fortran --enable-parallel --enable-shared
 make check
 sudo -E make install
cd ..

# note that netcdf fortran library is now compiled from a
# seperate source from the main netcdf c library. Install
# the c library first, and make sure to create the shared
# object file.
wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4.3.3.tar.gz
tar -xzf netcdf-4.3.3.tar.gz
cd netcdf-4.3.3/
 LIBS=-ldl CC=mpicc CXX=mpiCC FC=mpif90 CPPFLAGS=-I/usr/local/include/ LDFLAGS=-L/usr/local/lib/ ./
     ↪ configure --prefix=/usr/local --enable-parallel
 make check
 sudo -E make install
cd ..

wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-fortran-4.4.2.tar.gz
tar -xzf netcdf-fortran-4.4.2.tar.gz
cd netcdf-fortran-4.4.2
 LIBS=-ldl CC=mpicc CXX=mpiCC FC=mpif90 LDFLAGS=-L/usr/local/lib/ CPPFLAGS="-I/usr/local/include -
     ↪ DUSE_NETCDF4" ./configure --prefix=/usr/local
 make check
```

```
 sudo -E make install
cd ..

# convenient viewer for contents of netcdf files (not essential)
sudo apt-get install ncview

# ESMF requires ESMF_DIR and probably other environment variables.
# These can be set at the command line or, for example, in your
# .bashrc fileor a local script file. These might work:
export ESMF_DIR="/top/level/directory/for/esmf/"
export ESMF_NETCDF="split"
export ESMF_NETCDF_INCLUDE="/usr/local/include"
export ESMF_NETCDF_LIBPATH="/usr/local/lib"
export ESMF_COMM="openmpi"
export ESMF_PIO="internal"

wget downloads.sourceforge.net/project/esmf/ESMF_6_3_0r/ESMF_6_3_0rp1/esmf_6_3_0rp1_src.tar.gz
tar -xf esmf_6_3_0rp1_src.tar.gz
cd esmf
 make check
 sudo -E make install
cd ..

# In order to actually use ESMF you must set the environment
# variable ESMFMKFILE. If you didn't use environment
# variables to specify the install location this make file
# will probably end up somewhere like this:
export ESMFMKFILE="$ESMF_DIR/DEFAULTINSTALLDIR/lib/libO/Linux.gfortran.64.openmpi.default/esmf.mk"
```

... or for ESMF version 7.0.0 ...

```
wget https://sourceforge.net/projects/esmf/files/ESMF_7_0_0/esmf_7_0_0_src.tar.gz
tar -xf esmf_7_0_0_src.tar.gz
```

... or for ESMF version 7.1.0 beta snapshot 14 ...

```
git archive --remote=git://git.code.sf.net/p/esmf/esmf --format=tar --prefix=esmf/
    ↪ ESMF_7_1_0_beta_snapshot_14 | tar xf -
```